

<u>Cours de Génie Logiciel</u>

Introduction à la conduite de projet

Laurent Henocque
Maître de Conférences

Ecole Supérieure d'Ingénieurs de Luminy

Université de la Méditerranée Aix-Marseille II

Cours de génie logiciel
Introduction à la conduite de projet informatique

ESIL / ES2I

Université de la Méditerranée

par Laurent Henocque

(henocque@esil.univ-mrs.fr)
(http://www.esil.univ-mrs.fr/Staff/Prof/henocque/)

version 1.2 en date du 10 octobre 1996

1. le cycle de vie logiciel

On appelle cycle de vie d'un produit le parcours entier de ce produit depuis son étude préliminaire jusqu'à son abandon définitif. Le cycle de vie comporte donc une période assez longue durant laquelle le produit est exploité ou vendu et fait l'objet de maintenance et de développements complémentaires. Le cycle de vie débute par une phase de développement, qui se déroule de l'étude préliminaire à la livraison au client (ou « recette ») et qui nous intéresse ici.

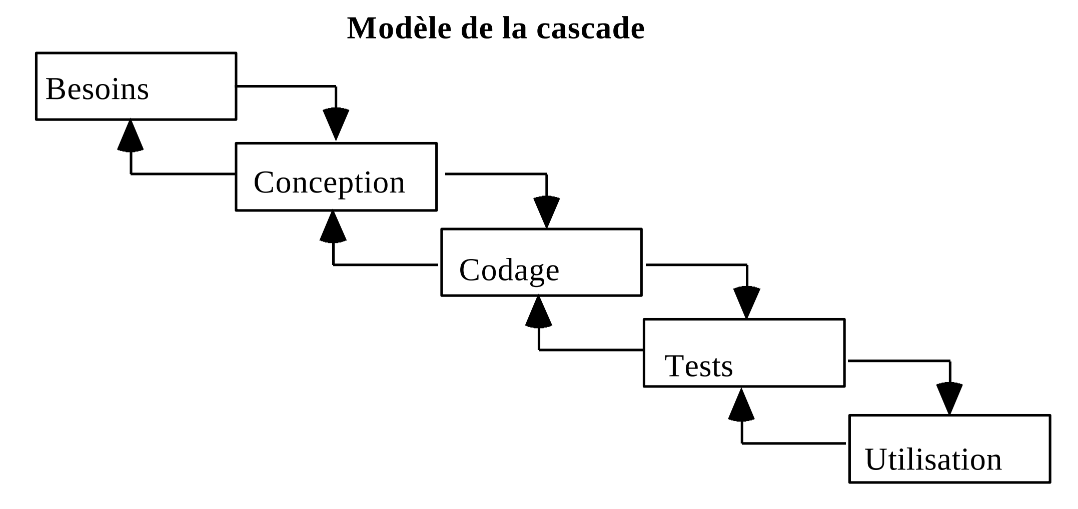
2. le processus de développement logiciel

2.1. le développement en cascade ou en V

2.1.1. la cascade simple

Le modèle le plus ancien de processus logiciel est hérité de la grande industrie et du bâtiment. Il consiste à considérer que le développement résulte d'un enchaînement de phases indépendantes, dont chacune doit être terminée pour pouvoir aborder la suivante. Bien que fondamentalement en désaccord avec la nature de l'activité de développement, ce modèle fournit une base très utilisée du fait qu'il permet une gestion analytique détaillée et un suivi méticuleux du projet, avec rédaction de documents administratifs à l'issue de chaque phase.

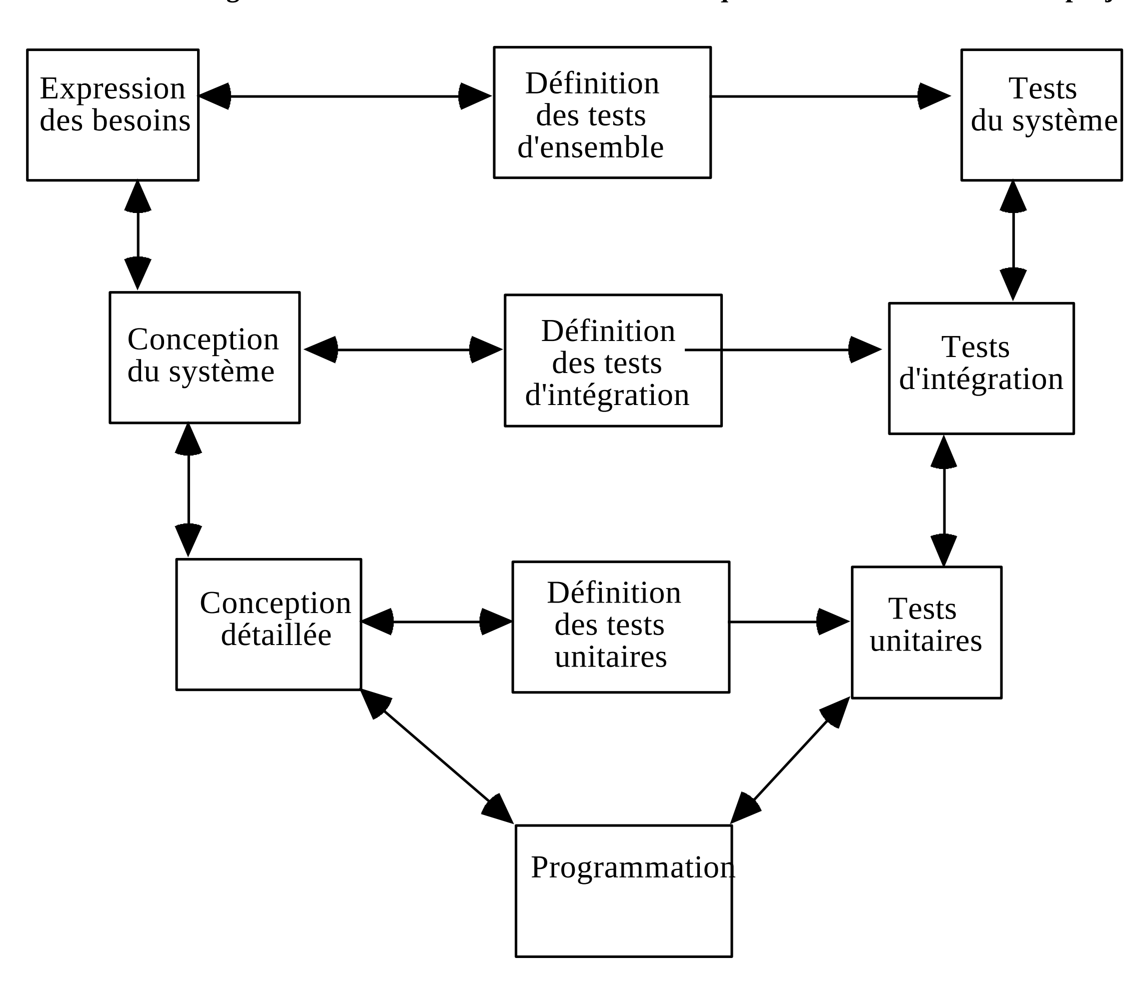
Modèle de la cascade



2.1.2. le processus de développementen V

Un modèle dérivé du précédent prend en compte les activités liées à la définition des tests. En effet, chaque phase du modèle de la cascade permet de définir certains types de tests, et modélise ainsi non seulement une succession temporelle d'évÈnements, mais aussi des niveaux variables d'observation du système.

La figure suivante illustre ce modèle :



Les principales phases retenues dans le processus logiciel sont les suivantes :

- la définition et l'analyse des besoins (spécification)
- la conception du système et du logiciel
- l'implantation et tests unitaires
- l'intégration et les tests système
- la mise en oeuvre et maintenance

La production de documentation à l'issue de chaque phase, et l'impératif absolu de maintenir la consistance de cette documentation avec le produit réellement fabriqué, conduit à une lourdeur des allers retours entre les phases. Il est pourtant fréquent que des considérations de codage révèlent des incomplétudes, ou des inconsistances dans le produit des phases supérieures. On doit souvent, pour éviter des aller retours trop nombreux et coûteux procéder au gel arbitraire de phases, en considérant par exemple que le produit de la spécification est définitivement acquis, et ne pourra pas être remis en cause. De telles interventions simulent ce qui se produit naturellement dans le bâtiment par exemple, et montrent bien la différence fondamentale qui existe entre l'activité informatique et la plupart des autres activités humaines.

Le modèle de la cascade facilite notamment la comptabilité des temps de développement, et permet de produire la documentation officielle du produit qui sert aux deux opérations de vérification et de validation :

- vérification : construisons nous le produit tel qu'il est spécifié?
- validation: construisons nous le produit qui correspond aux besoins réels du client?

2.2. la programmation exploratoire

La programmation exploratoire est modification continue et itérative d'un prototype jusqu'à l'obtention du produit final. Elle correspond souvent à des applications d'intelligence artificielle, réalisées avec des langages de haut niveau (prolog, lisp), et sert dans les cas où il semble impossible de réaliser une spécification détaillée du produit.

La confrontation périodique du prototype avec le client, et la simulation de son comportement sur des jeux d'essais fournis par le client, permettent de se rapprocher du but final. Une telle approche aboutit souvent à une réalisation qui se révèle à l'usage incomplète, c'est à dire ne couvrant qu'un partie du domaine à priori immense dévoilé par le client. Elle est rarement utilisée pour systèmes autres que basés sur l' IA , car des activités de gestion rigoureuse ne sont pas faciles à mettre en oeuvre(le système change souvent).

2.3. le prototypage

Le prototype sert uniquement à préciser les besoins. Il associe le client à un processus de raffinement progressif du prototype, à l'issue duquel les besoins sont parfaitement définis pour les deux contractants. Le prototypage est beaucoup utilisé pour spécifier les interfaces homme machine, pour lesquelles on dispose d'outils de génération de code qui permettent des cycles d'interaction très brefs. Certains outils permettent même de concevoir l'interface en direct, à côté du client.

La pratique du prototypage s'étend progressivement à tous les domaines pour lesquels on dispose de générateurs de code, l'objectif étant de satisfaire au mieux les besoins du client, en évitant tout écart entre les besoins réels (needs) et les besoins exprimés (requirements) par ce dernier. La tendance actuelle est de mettre totalement en sourdine les velléités des équipes de développement à réaliser des programmes technologiquement “intéressants” **-** cela s'appelle “technology push” **-** , pour favoriser une approche où domine l'écoute totale du client **-** nommée “social pull” **-**.

2.4. la transformation formelle

La transformation formelle est un processus par lequel on obtient la dérivation automatique du code exécutable à partir de sa spécification, ou de sa conception. Les outils qui le permettent parlent de spécifications exécutables, ou de conception exécutable.

De nombreux outils de génération d'interfaces homme machine proposent partiellement cette fonctionnalité. Des sociétés (software through pictures par exemple) proposent des outils qui permettent de générer le code exécutable à partir de spécifications reposant sur les diagrammes de flots de données et de flots de contrôle du type SADT. La société CGI a ainsi construit son image de leader dans le domaine du service grâce à un produit interne appelé PacBase qui permet de développer un application (presque) devant le client.

2.5. l'assemblage de composants réutilisables

Ce processus consiste à combiner des briques élémentaires au sein d'un ensemble fonctionnel, comme une construction de legos par exemple. L'idée est d'éviter toute recompilation, sinon on se situe au niveau apporté par la programmation objet. On se place donc au plus haut niveau possible sur la machine (processus), on normalise les communications entre processus[1], on organise la communication et l'accès aux services.

L'exemple le plus fameux de cette approche est le système Unix, qui propose un système unifié de communication par des fichiers, et la possibilité de combiner des commandes entre elles pour parvenir à un résultat. Aujourd'hui, les environnements de contrôle proposés pour l'exécution d'applications distribuées étendent le même type de concepts au niveau d'un réseau hétérogène. C'est le cas de DCE (distributed computing environnement) de l'OSF, et de l'UNO (universal networked objects) de SunSoft. Un modèle comme PCTE apporte aussi ce type de fonctionnalité en permettant de combiner des éléments logiciels pour réaliser un AGL. PCTE comporte un système globale de gestion de données (repository) reposant sur le modèle entité relation association.

3. la gestion de projet

3.1. les modèles de gestion de projet

3.1.1. les modèles basés sur les délivrables

Le processus est divisé en étapes indépendantes, consécutives ou non, et chaque étape donne lieu à une revue et produit un document

Voici par exemple une liste possible de documents pour le modèle de la cascade :

Phase/Activité	Document
Etude préliminaire des besoins	rapport sur la faisabilité
	grandes lignes des besoins
Spécification des besoins	spécification des besoins
	(cahier des charges)
Spécification détaillée	spécification fonctionnelle
	spécification des tests d'acceptation

[1]entrées et sorties doivent être compatibles

	manuel d'utilisation préliminaire
	(cahier des charges fonctionnel)
Modélisation du système	spécification de l'architecture
	spécification des tests du système
Conception des interfaces	spécification des interfaces
	spécification des tests d'intégration
Conception détaillée	spécification de la conception
	spécification des tests unitaires
Codage	programme
Tests unitaires	rapports
Tests de modules	rapports
Tests d'intégration	rapports
	manuel utilisateur final
Tests du système	rapport
Acceptation	système final

3.1.1.1.problèmes liés à ce modèle

Le calendrier de délivrance de documents peut ne pas être en phase avec le calendrier de l'avancement technique, et produire dans ce cas des documents inadéquats.

Le coût de mise à jour des délivrables peut devenir prohibitif lors d'itérations (i.e. retour à une phase antérieure) et conduire à l'abandon de modifications du système au profit de solutions inélégantes dans le seul but de maintenir la consistance des documents avec le produit.

La structure linéaire de production des documents est artificielle. On ne spécifie pas un produit qu' on saurait ne pas être capable de réaliser. Un processus basé sur les documents peut créer des masques sur l'information sources de difficultés futures.

Approuver un document est long et difficile. En général, on démarre une phase avant que les documents de la phase précédente soient terminés.

On ne peut résoudre tous les problèmes en appliquant scrupuleusement un tel modèle, simplement parce que les aspects temporels (des phases du projet) sont sans doute moins importants que les aspects comportementaux : cognitifs, sociaux et humains. Les individus sont victorieux dans leurs projets justement quand ils parviennent à franchir les barrières administratives.

3.1.2. un modèle basé sur le risque : la spirale de Boehm

Le modèle de la spirale du risque de Boehm met en oeuvre une évaluation régulière des risques liés au projet, permettant la mise en oeuvre de solutions techniques pour annihiler ou contrer ces risques. Elle englobe les autres approches, en permettant qu'un cycle de la spirale utilise un modèle de développement en cascade (quand le risque est un risque d'intégration par exemple), ou bien utilise le prototypage quand le risque est lié à l'acceptation de l'interface utilisateur par le client. A ce titre, c'est un méta modèle.

Chaque cycle de la spirale considère les six phases suivantes :

1 déterminer les objectifs, les alternatives et les contraintes

2 évaluer les alternatives, identifier et résoudre les risques

3 réaliser un éventuel prototype

4 faire des simulations, des modèles de mise en oeuvre et des tests de performances

5 développer et vérifier le produit qui servira de base au prochain niveau

6 planifier la prochaine phase

Chaque cycle de la spirale donne lieu à la rédaction d'un document qui renseigne les rubriques suivantes :

objectifs

contraintes

alternatives

risques

résolution des risques

résultats

plans

conséquences pour la suite du projet

3.1.2.1.Exemple de spirale complète d'un projet logiciel

3.1.2.1.1.cycle 1

1. revue de départ,
2. risque,
3. prototype 1,
4. simulation,
5. concept de l'opération,
6. planification des besoins et du cycle de vie

3.1.2.1.2.cycle 2

1. objectifs,
2. risques,
3. prototype 2,
4. simulation,
5. besoins logiciels, validation des besoins
6. plan de développement

3.1.2.1.3.cycle 3

1. objectifs,
2. risques,

3 prototype 3,
4 simulation,
5 conception du produit, validation et vérification de la conception
6 plan de test et d'intégration

3.1.2.1.4.cycle 4

1 objectifs,
2 risques,
3 prototype opérationnel,
4 simulation,
5 conception détaillée, code, tests unitaires, d'intégration, d'acceptation, mise en service
6 planification de la phase suivante

3.1.3. le cas particulier du développement rapide d’applications (RAD)

Une utilisation prometteuse du modèle de la spirale de Boehm est utilisée aujourd’hui par des projets de moyenne importance. Elle consiste à imposer une durée constante, ou quasi constante, aux cycles successifs du modèle. Cette approche incorpore la gestion du risque dans un modèle de type prototypage, et possède plusieurs avantages :

- elle donne au client une vision du produit en cours de développement à intervalles réguliers, et suffisamment tôt pour permettre une réaction utile en présence d’une impossibilité manifeste de réalisation, ou d’une mauvaise interprétation des besoins. On dit que cela annule l’effet « tunnel » par lequel un projet est invisible pendant tout son développement, pour finalement apparaître le jour de la recette.
- elle oblige à des choix techniques spécifiques dans la mesure où chaque cycle, dans un délai assez bref, doit générer un produit « démontrable ».
- elle oblige à mettre en place d’emblée toute la chaîne de solutions techniques nécessaires à permettre cette démontrabilité. Ainsi, des problèmes éventuels sont décelés plus tôt.
- elle place sur les équipes une pression quasi constante (on est toujours à j-n de la fin d’un cycle), évitant ainsi une montée en charge exponentielle et catastrophique sur la fin du projet. Cela améliore également la productivité.

3.2. la gestion de projet

Le chef de projet rend compte à un directeur informatique. Il encadre une équipe d'ingénieurs pouvant comporter de cinq à une centaine d'individus. Il rend possible le travail de ses équipes, et à ce titre il organise les structures de développement, l'infrastructure matérielle, la gestion des hommes. D'un point de vue plus "mesurable", il est responsable :

- de la planification du projet,
- des coûts,

- du respect des normes, en relation avec l'assurance qualité
- des délais

Les difficultés sont de deux ordres : celles liées à la gestion des individus, et celles liées à la technique. Au plan humain, le chef de projet est en charge d'individus ayant un très fort niveau de formation (niveau maîtrise très souvent), et une attitude d'esprit assez indépendante, renforcée par la pratique isolée de l'activité de programmation[2]. Ces caractéristiques rendent les frondes difficiles à gérer, et mettent toujours le chef de projet en situation délicate lorsqu'un désaccord apparaît entre lui et le chef d'équipe technique, qu'il y a tout lieu de choisir comme l'individu le plus apte du groupe à un moment donné. Au plan technique, le chef de projet informatique exerce une activité de gestion dans un domaine immatériel, le logiciel, dont l'état d'avancement est bien moins facile à observer que celui d'un bâtiment en construction. Par ailleurs, de nombreux projets informatiques constituent des innovations, car on ne connaît aucun précédent semblable au projet couramment à réaliser. Enfin, le processus de développement du logiciel est peu connu, la preuve en étant que certains aspects apparemment non techniques : sociaux, liés à la communication, à la formation, jouent un rôle prépondérant dans la productivité et la qualité du résultat final.

Le chef de projet informatique exerce donc une activité qui, bien qu'essentiellement administrative, laisse une grande part à l'improvisation et aux qualités individuelles.

3.2.1. les activités de gestion

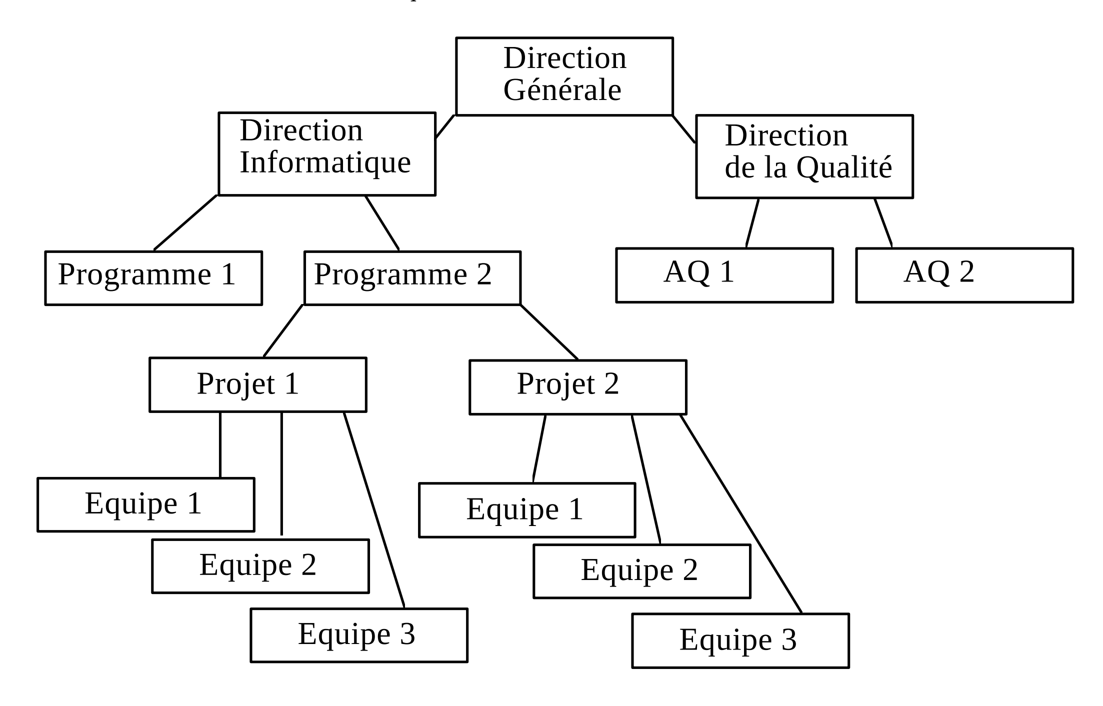
Le chef de projet organise un ensemble d'activités externes au développement proprement dit, afin de maîtriser le processus de développement, et de garantir la tenue de ses engagements financiers et techniques. Maîtriser le processus de développement revient à se donner les moyens de connaître l'état d'avancement du projet, au moyen de revues périodiques dont les résultats sont mesurables. Les activités du chef de projet sont les suivantes :

- la rédaction de la proposition technique initiale (en réponse par exemple à une offre de marché public du BOAMP)
- le chiffrage du projet
- la planification, l'ordonnancement du projet, et l'allocation des ressources aux différentes tâches à réaliser
- le suivi informel et les revues formelles
- le recrutement et l'évaluation du personnel. Il faut qu'au moins un membre du projet ait l'expérience d'un projet comparable
- la rédaction et la présentation de rapports
- la communication avec le client et avec le directeur informatique
- la fourniture aux équipes des moyens techniques de leur activité : matériels, formation, salaires ...

[2]voir à ce titre la partie qui traite de la psychologie du programmeur dans ce chapître.

3.2.2. les structures de gestion de projet

La hiérarchie habituelle des services informatiques dans les sociétés est la suivante :



Pour des raisons évidentes d'indépendance, la direction de la qualité n'est pas soumise hiérarchiquement à la direction informatique. Cela s'explique aussi parce que le suivi de la qualité lors de la réalisation de systèmes industriels ne s'applique pas aux seules activités informatiques.

3.2.2.1.l'organisation des équipes de programmation

Les équipes de programmation gagnent à être constituées de deux à cinq personnes, socialement et humainement compatibles, techniquement assez homogènes, ou alors dont les différences entre les membres sont acceptées comme une complémentarité. Lorsque la spécification détaillée du système informatique a été réalisée, les modules sont identifiés, et leurs interfaces bien définies. Il est alors possible de répartir la charge de travail sur plusieurs équipes autonomes, et indépendantes, ayant vocation de réaliser des modules conformément à leur spécification. Les avantages de la petite équipe sont les suivants :

- il peut être défini un standard de qualité interne à l'équipe, celle ci disposant d'une relative autonomie pourvu qu'elle atteigne ses objectifs
- il peut se développer une intimité entre les membres de l'équipe, qui améliore la qualité de la communication, et de fait la productivité
- une petite équipe peut pratiquer la programmation impersonnelle, c'est à dire globaliser la responsabilité des erreurs et le mérite des succès, ce qui est aussi reconnu comme un gage de productivité
- on peut y avoir une meilleure connaissance des activités de chacun
- il y a possibilité de fonctionnement consensuel pour les décisions d'ordre technique comme les attributions de tâches

- on économise une phase (souvent la plus délicate) du modèle de la cascade : la conception de bas niveau peut être laissée aux individus, et ne demande donc pas d'être traitée de façon administrative. Seule la conception de haut niveau reste confiée à un membre senior

3.2.3. la productivité du programmeur

La productivité du programmeur est une donnée essentielle de la gestion de projet. Elle détermine la planification, et elle conditionne l'atteinte des objectifs. Pourtant, elle est extrêmement difficile à mesurer. Les métriques connues de productivité reviennent toujours peu ou prou à compter le nombre de lignes de programme écrites par une personne, et permettent ainsi de détecter le tire au flan magnifique, avec cependant les inconvénients suivants :

- on devrait se rapporter à des lignes écrites en langage machine (celles que génère un compilateur) car la taille de programmes sources varie considérablement en fonction du langage utilisé (Lisp, Prolog vs C par exemple)

- on favorise le programmeur qui écrit des programmes inutilement longs, et parfois stupidement longs, au détriment de celui qui réfléchit avant de programmer,

- on n'encourage pas la concision du code, et partant sa maintenabilité et sa qualité

- on ne tient pas compte des commentaires et de la documentation

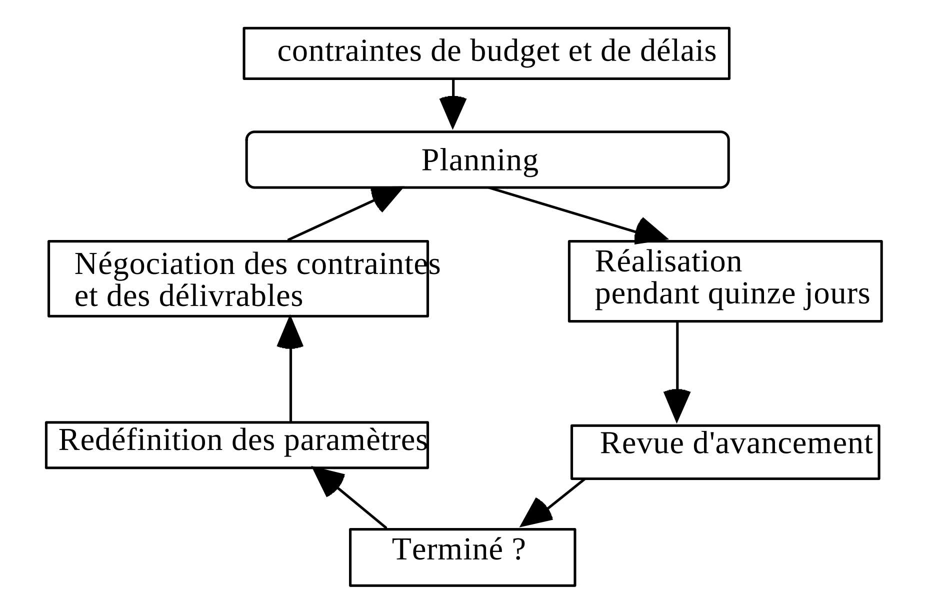
- la réutilisation de composants existants est un facteur de qualité

Certains auteurs proposent de mesurer les volumes de documentation produite, et les lignes de commentaires. Sans qu'il soit possible de véritablement trancher sur le sujet, disons que la productivité du programmeur ne peut se mesurer que si on corrèle production et qualité. On verra plus loin des propositions de métriques de qualité des programmes, qui donnent d'autres éléments de réflexion sur ce thème.

Pour conclure, disons qu'un chef de projet est toujours dans une situation délicate s'il a l'intuition qu'un de ses développeurs ne produit pas au même rythme que les autres, car les arguments du paresseux, de l'incompétent, ou de l'individu de mauvaise foi sont difficiles à contredire. Néanmoins, il est clair que si un tel doute persiste, l'éviction ou la réorganisation des équipes est salutaire au reste de l'équipe et du projet.

3.3. la planification

Le chef de projet organise dans le temps les activités de ses équipes. Le processus de cette planification est dynamique, devant être en permanence reconsidéré en fonction de la situation réelle, à mesure que des informations nouvelles sont acquises.



Une difficulté, liée au développement, réside dans le fait que la venue de problèmes techniques pose sur le projet des contraintes qui sont contradictoires avec les contraintes initiales. Reconsidérer le planning comporte dans ce cas une phase de négociation avec le client du projet, pour accord sur d'éventuelles modifications des besoins liées à la situation concrète. Le chef de projet a donc avantage à se protéger, dans des marges raisonnables, en ne promettant pas des délais où des coûts trop serrés, mais surtout en assortissant ses engagements de conditions portant sur les conditions de travail : achat d'outils, vitesse des machines, performance des compilateurs. Ainsi, sa responsabilité est engagée de façon moindre lorsqu'un dépassement de coûts ou de délais ne relève pas de lui mais d'une défaillance dans des fournitures matérielles.

Un chef de projet doit savoir dire : "Non! Je ne peux réaliser ce projet dans de tels délais, avec de tels coûts et sous telles contraintes."

Notons que le chef de projet planifie non seulement le développement, mais également les revues de validation, la gestion des configurations, la formation, la maintenance

3.3.1. les revues du projet

Les délivrables, c'est à dire documentation, rapports et exécutables, sont les seuls moyens par lesquels on peut véritablement décider de l'avancement d'un projet. Le chef de projet doit mettre en oeuvre un processus de suivi et de revues régulières dont les échéances dans le temps doivent être assez rapprochées pour permettre des corrections précoces. Le projet est donc découpé en activités élémentaires dont les critères de bonne fin sont aisément définissables et testables, et des revues sont organisées toutes les deux à trois semaines.

Une échéance n'est validée que si les conditions suivantes sont réunies :

- la documentation est terminée
- l'avancement est mesurable dans l'absolu (i.e. terminé vs non terminé), mais pas sous forme de pourcentage toujours subjectif (interface utilisateur à moitié terminée). (Un projet qui ne marche pas est toujours fini à 90%).
- un rapport sur l'avancement peut être rédigé et envoyé à la direction

Le chef de projet, en planifiant les échéances, doit définir également des critères de transition entre phases, c'est à dire des conditions devant être effectivement atteintes pour pouvoir passer à la phase suivante.

3.3.2. la planification

Le chef de projet détermine, en accord avec ses chefs d'équipe, les temps de développement associés à toutes les activités. Pour le faire, il doit prendre en compte des imprévus inhérents au développement. En fait, quand on aborde le problème par en haut, ce qui est le cas lors des planifications initiales, on envisage pour toutes les activités des problèmes potentiels. Pour prévoir la durée d'une activité, on calcule sa durée dans le meilleur des cas, puis on y ajoute un délai de garantie des problèmes envisagés, puis on pondère le tout pour tenir compte de l'imprévu.

La gestion des ressources et du temps se fait aujourd'hui à l'aide d'outils informatiques adaptés, dits logiciels de gestion de projet. Ces programmes prennent en compte notamment:

- les contraintes temporelles entre activités
- l'association des ressources aux activités
- le calcul des coûts afférents au projet sur la base de durées et de coûts unitaires
- des calendriers individuels de disponibilité des ressources

Notons toutefois que peu de ces produits s'avèrent capables de calculer par exemple une affectation des ressources aux activités qui optimiserait un critère donné (délai minimum de réalisation, coût minimal, etc...). Egalement, aucun produit ne permet de conditionner la durée d'une activité à la personne qui la prend en charge (un spécialiste réseau mettre moins de temps sur certaines activités qu'un débutant). Ces décision sont donc laissées à la discrétion du chef de projet, qui devra en plus prendre en compte des impératifs sociaux…

Trois diagrammes fondamentaux permettent de représenter les éléments de gestion des ressources et du temps d'un projet.

3.3.2.1.diagramme Pert

Le diagramme d'ordonnancement des activités, ou PERT, est un graphe ordonné décrivant les contraintes de précédence logique des activités. La spécification doit avoir lieu avant la conception, et cette dernière avant le codage, par exemple.

Ce diagramme permet de décrire graphiquement ces contraintes, qui complétées par des informations sur la durée des activités élémentaires, permet de calculer pour chaque activité quel est son début au plus tôt, sa fin au plus tôt, son début au plus tard, et sa fin au plus tard.

Le graphe de PERT permet de visualiser ce qu'on appelle le chemin critique, c'est à dire le chemin conduisant du début à la fin du projet dont la durée minimale est maximale. C'est ce chemin qui conditionne la fin au plus tôt du projet. Toute variation de durée d'une des activités critiques se répercute donc sur le projet global, ce qui leur vaut un suivi tout particulier du chef de projet.

3.3.2.2.diagramme de Gantt

Ce diagramme figure chaque activité sur un calendrier sous forme d'une barre grisée débutant à la date de début au plus tôt et terminant à la date de fin au plus tard, sur laquelle glisse une barre blanche correspondant aux dates réelles de début et de fin.

Comme les efforts sont généralement déterminés en jours * homme, le glissement de la barre blanche d'exercice réel tient compte des calendriers de disponibilité des ressources affectées à cette activité.

3.3.2.3.diagramme d'emploi des ressources

Ce dernier diagramme illustre, selon le même format que le diagramme de gantt, quel est l'emploi du temps de chaque ressource.

3.4. l'estimation des coûts

L'estimation d'un coût se fait en présence d'une définition fixe des besoins, même si elle est sommaire . Boehm rapporte les méthodes suivantes d'évaluation des coûts :

- la modélisation algorithmique des coûts : calcul réalisé à partir de différents critères numériques : taille, coût d'un logiciel comparable…
- le jugement contradictoire d'experts, ramené à un coût unique par consensus ou par moyenne
- l'estimation par analogie : lorsqu'on a déjà réalisé un logiciel comparable
- la loi de parkinson : on utilise toujours le temps accordé, quels que soient les objectifs. Le coût est donc égal à l'effort que l'on peut fournir.
- le coût objectif , ou encore le prix gagnant : c'est le prix que le client est disposé à payer pour son produit.
- l'estimation descendante, basée sur les fonctionnalités à produire, plutôt que sur les coûts des éléments à produire pour implanter ces fonctions
- l'estimation ascendante : cumuls des coûts des composants

Ces méthodes se valent toutes. Le choix de l'une ou l'autre repose sur des considérations liées au contexte. En général, le coût total est évalué avec une combinaison de ces approches.

Disons toutefois que l'estimation ascendante donne souvent des résultats catastrophiques si les coûts élémentaires sont surévalués, que l'estimation descendante ne peut être réalisée que par des chefs de projets confirmés, que le coût objectif est très employé en pratique[3], que la loi de parkinson est véridique[4], que l'estimation analogique est précieuse, que le jugement d'experts dit aussi estimation "à la louche" est notoirement fiable, et enfin que les modèles algorithmiques de calcul de coût sont difficiles à mettre en oeuvre à un petit niveau, lourds, et que leurs résultats sont de toutes façons confrontés aux modes précédents d'estimation.

Dans la pratique, une bonne manière de s'assurer qu'un coût est réaliste consiste à faire converger les estimations provenant de différentes sources. Une dernière technique très efficace de jugement contradictoire d'experts appelée la méthode Delphi permet d'atteindre des estimations réalistes et précises. Voir sur ce point le cours CMM.

---

[3]le client et le prestataire, une fois d'accord sur le prix de cession, definissent les fonctionnalités à prix constant.

[4]même sous des contraintes de délais énormes, les programmeurs déploient des trésors d'imagination pour atteindre les objectifs en temps quasiment voulu.

3.4.1. le choix des options alternatives

Un projet informatique laisse ouvertes en général de nombreuses alternatives de réalisation. Le coût d'une solution est évidemment un critère pour choisir entre différentes possibilités. Il existe de nombreuses approches pour le choix d'options basées sur le coût, dont une est normalisée sous le nom d'analyse de la valeur, et fait l'objet d'un chapitre de ce cours. Nous abordons ici plusieurs méthodes élémentaires.

Chaque solution peut être qualifiée d'un coût minimal, et d'un coût maximal. On peut choisir la solution dite "minimin", dont le coût minimal est le plus faible, la solution dite "minimax" (ou encore "de moindre regret"), dont le coût maximal est le plus faible, ce qui minimise le risque, la solution de coût moyen minimal (on considère que la distribution des coûts) est équiprobable, ou la solution qui minimise un coût probable, calculé en tenant compte des espérances d'atteindre, pour chaque solution, le coût minimal et le coût maximal.

L'objectif est ici de minimiser le risque, mais aussi de déterminer le choix d'une solution en tenant compte non seulement de son coût, mais de sa valeur qui peut être vue comme son rapport qualité prix. C'est l'intérêt de l'analyse de la valeur. En effet, deux solutions différentes pourront avoir des intérêts différents pour le prestataire (internes et indépendants du projet lui même), parce que l'une permet la formation d'un débutant et pas l'autre, ou parce que l'une permet la réalisation d'outils qui seront réutilisés dans d'autres projets, ou commercialisés sous forme de produit. On voit donc qu'indépendamment du prix de cession qui est l'objet d'un contrat avec le client, le coût n'est pas nécessairement le critère fondamental de choix d'une solution. Une organisation peut décider d'investir des fonds propres sur un projet, qui n'est donc que partiellement financé par le client, pour des raisons stratégiques.

Pour conclure, le facteur risque domine le processus de décision, dans la mesure ou il est nÈcessaire que le projet aboutisse. Sommerville écrit :"le succès est toujours meilleur marché que l'échec!".

3.4.2. la prévision de la taille des programmes

On connaît le coût de projets terminés, dont on connaît la également la taille, exprimée en KISL : nombre de Kilo Instructions Sources Livrées) . Il est légitime de penser utiliser des statistiques sur ce qui est connu, pour permettre de prédire le coût d'un projet dont on saurait prédire la taille.

On veut établir une corrélation prédictive, au moyen d'une expression mathématique, entre des attributs du programme qui doit être réalisé et l'effort à fournit pour atteindre les objectifs. La première étape consiste à corréler les fonctionnalités du produit et la taille du résultat. Les attributs doivent être estimés avant le codage, et relèvent par exemple des entrées sorties, les interactions avec l'utilisateur, les interfaces externes, les fichiers utilisés par le système… Albrecht et Gaffney déterminent sur cette base des "points fonction", calculés en donnant un poids particulier à chaque type d'attribut, et à chaque attribut suivant son poids propre (un fichier très long coûte plus qu'un court) Les statistiques connues permettent alors d'écrire une formule du type :

taille du code = C * nombre de points fonction

C est une constante décrivant l'expansion des points fonction en nombre de KISL en fonction du langage de programmation utilisé.

3.4.3. le calcul de l'effort à partir de la taille

On observe statistiquement que le coût d'un logiciel est une fonction exponentielle de sa taille, selon une formule du type :

effort(HM) = A * (KISL)^B

Les coefficients A et B varient en fonction du type de projet informatique :

- organique équipes petites et intimes travaillant sur un domaine bien compris
- intégré développement de logiciels intégrés à des systèmes industriels complexes (aviation, espace)
- semi détaché type intermédiaire entre les deux précédents

Voici des exemples de valeurs des coefficients :

```
mode organique          HM = 2,4 * KISL^1,05

mode semi détaché HM = 3  *  KISL^1,12

mode intégré            HM = 3,6 * KISL^1,20
```

Par ailleurs, on dispose de formules pour calculer le temps de développement du projet, connaissant l'effort à fournir, en supposant toujours disposer de toutes les ressources nécessaires :

```
mode organique          TDEV = 2,5 * HM^0,38

mode semi détaché TDEV = 2,5 * HM^0,35

mode intégré            TDEV = 2,5 * HM^0,32
```

Pour 32 KISL, en mode organique, on doit tabler sur une effort de 91 HM, et un temps de développement de 14 mois. Pour 128 KISL, en mode intégré, il faudra prévoir 1216 HM, et une temps de développement de 24 mois.

3.5. la conduite du groupe

3.5.1. le chef technique contre le chef administratif

Un projet informatique nécessite toujours la mise en place d’une autorité administrative. Ce directeur est responsable aux yeux de ses supérieurs de la bonne marche du projet, en termes de coût global, et exerce donc toutes les activités de gestion associées au projet. Ce chef n’est en général pas celui que se donne l’équipe, qui est tentée de se ranger à l’avis du plus influent, le plus souvent celui dont l’autorité technique est patente, ou indiscutée. La raison en est que les problèmes qui se posent à une équipe de programmeurs ont toujours une projection au niveau technique : la bonne méthode, le bon outil, la connaissance d’algorithmes ou d’une procédure pour résoudre un problème seront en général le sésame de programmeurs dont le projet dérape dans le temps.

Au long du projet, le chef technique peut changer, alors que le chef administratif est en général conservé, même lors de grandes difficultés, car le coût potentiel induit par l'incorporation d'un élément nouveau dans un projet technique traduit un risque qui n'est pas souvent pris.

On constate que l'autorité technique et l'autorité administrative gagnent à être dissociées. En effet, les charges administratives constituent un fardeau qui gâcherait les ressources du technicien. Réciproquement, on constate que lors de périodes techniquement difficiles, les programmeurs, dont le chef technique doit faire partie, ont tendance à se couper totalement du monde. Certaines responsabilités administratives simples deviennent alors oubliées[5] et se traduisent à terme par des difficultés.

La productivité d'une équipe qui contient les hommes compétents dans les domaines à couvrir par le projet est considérablement supérieure à celle d'une équipe où manquent de telles compétences. Il est donc nécessaire de conserver au niveau technique des individus dont le savoir faire est reconnu, sans pour autant qu'ils ne se sentent lésés, en terme d'avancement et de carrière, par rapport à leurs collègues non techniciens. Le développeur doué a en général toujours le sentiment de faire vivre la société dans laquelle il travaille lorsque sa production est commercialisée, et veut en être payé. Pour résoudre ce problème, IBM a défini ce qu'il est convenu d'appeler l'équipe du chef programmeur. Cette équipe est organisée autour d'un individu très compétent **-** le chef programmeur **-**, et se compose d'un assistant, qui décharge le premier de certaines activités de programmation bien spécifiées, d'une secrétaire qui règle tous les détails administratifs du projet, et fait intervenir au besoin ponctuellement des experts dans des domaines utiles au projet. Cette structure permet notamment de résoudre les difficultés suivantes :

- pas de conflits de personnalité,
- réduction des charges administratives par une équipe minimale, très performante, dont on peut attendre que le planning soit respecté, du fait d'une longue expérience,
- positionnement du chef programmeur à une niveau hiérarchique élevé, compatible avec un salaire également élevé.

3.5.2. le travail en groupe

3.5.2.1.les comportements de groupe

3.5.2.1.1.fidélité au groupe

On constate que dans un groupe bien dirigé, dont les différentes autorités sont acceptées et dans laquelle la qualité de travail est réelle, les individus tendent à s'identifier au projet commun, à vivre globalement l'objectif poursuivi, et à faire corps lors de difficultés. Ce mécanisme, dit de “fidélité au groupe” possède des avantages, car on peut penser qu'il favorise une bonne communication entre ses membres, qui défendront chacun le groupe d'éventuelles influences (agressions) extérieures, mais il possède également des inconvénients, dont notamment :

- la résistance au changement de chef, notamment lorsque le projet est menacé par des défaillances techniques, ou un glissement trop important du planning,

[5] par exemple, les relations avec les fournisseurs et les clients.

- une diminution préjudiciable de l'esprit critique, la réalité d'une situation difficile étant abordée au travers d'un filtre de justifications plus ou moins réelles dont l'effet est de tenter de nier l'évidence.

3.5.2.1.2.la communication dans le groupe

On peut difficilement induire un modèle de communication dans un groupe. On constate en général un état de fait, qui lorsque tout va bien est sans importance[6] mais où, quand tout va mal montre à la fois des excès et des manques de communication.

On souhaite généralement réduire le niveau de communication improductive. Cela ne peut se faire que dans une mesure réduite car les individus ont besoin de jouer un rôle social au sein d'une structure. Dans de nombreuses cultures, les point importants ne peuvent être débattus qu'après une longue conversation informelle. Même lorsque ce n'est pas le cas, chacun a pu observer que parfois, un point très important est soulevé lors d'une discussion non productive, simplement parce qu'une idée vient de l'éclairer d'un jour nouveau ou simplement le faire resurgir. On ne peut donc ni qualifier ni quantifier la productivité de la communication. Tout au plus doit on veiller à ce que de véritables comportements de paresse ne s'installent.

Un groupe souffre en général d'avantage de manque de communication que de l'excès inverse. Il peut s'agir de la mise en quarantaine d'un individu, qui lorsqu'il est le chef administratif se trouve dans une position rendant sa gestion impossible. Le manque de communication entre individus est préjudiciable à un projet, car l'ensemble de l'activité informatique requiert de la malléabilité. Si le programmeur (et plus généralement l'utilisateur d'informatique) est habitué à s'adapter à l'état d'un système qu'il utilise, cette adaptation peut être plus difficile quand l'auteur du système ou de la norme d'utilisation est un membre du groupe avec lequel existe un conflit de personnalité. On voit ainsi des programmeurs faire dans leur coin une obstruction larvée à des consignes du groupe, augmentant la charge de travail de tous le jour où la défaillance est constatée, et manoeuvrant assez habilement pour ne pas toutefois pouvoir être vraiment inquiété.

Les conflits de personnes à la base de blocages dans un groupe peuvent, mais pas toujours, être dus à des personnalités trop fortes. On constate que les membres d'un groupe où des conflits existent portent souvent leurs griefs sur les quantités de travail fournies par les uns et les autres : "untel ne travaille pas assez". Il faut comprendre que dans un groupe important, il existe toujours des développements amont et des développements aval. On voit donc les programmeurs des seconds attendre celle des premiers. Si l'équipe amont ne respecte pas son planning, l'équipe aval lui fait grief de la qualité ou de la quantité de son travail. Respectivement, l'équipe amont qui se dit débordée, reprochera aux autres de se tourner les pouces et de ne pas faire le moindre effort pour les aider. On gagne donc en usure nerveuse dans l'équipe en permettant aux uns et aux autres de travailler aussi simultanément que possible, et de tester leurs modules indépendamment, à l'aide de simulateurs par exemple. Il n'existe en général pas de solution aux conflits de personnes, et une équipe où de tels problèmes surgit doit à terme être répartie sur des sous projets indépendants, où des projets distincts.

Notons que la généralisation des réseaux internationaux et d'entreprise permet à la quasi intégralité de la communication technique de se faire par ce biais. De ce fait, on peut restreindre la communication verbale à son rôle social, et clarifier ainsi la situation, en réduisant également le bruit dans l'enceinte de travail.

Une enquête réalisée aux états unis a montré que les équipes les plus productives sont constituées de deux ingénieurs, et que cette forme d'équipe concrétise son projet en six mois, pratiquement quelle que soit sa difficulté.

---

[6] l'euphorie d'un projet qui avance bien est parfois artificielle et peut conduire à des pertes de temps

3.5.2.2.la programmation impersonnelle

L'apparition de difficultés techniques ou matérielles dans un groupe est souvent génératrice de conflits inter individus qui mettent en cause leur production. On s'aperçoit alors que chacun est lié à ses programmes de façon affective, à tel point qu'aucun membre du groupe ne peut prendre la responsabilité de modifier, voire même de consulter les programmes d'un autre sans (risquer de) causer un drame. Il est donc nécessaire de réduire l'identification du programmeur à ses programmes, en assurant une mobilité dans l'élaboration des programmes. Cette mobilité ne peut être acquise que si un certain nombre de conditions sont réunies :

- **responsabilité** : l'ensemble de la production de l'équipe (des spécifications à la documentation et la maintenance) est réputée être sous la responsabilité du groupe tout entier, sans qu'on puisse imputer un bug à celui là seul qui l'a écrit,
- **questions** : il est fait obligation aux membres du groupe de toujours chercher à répondre par eux mêmes aux questions qu'ils pourraient avoir sur un programme écrit par un autre, en consultant les sources, ou toute forme de documentation liée au programme, avant de poser une question directe à son auteur présumé,
- **normes** : il est fait obligation aux membres du groupe de respecter les normes définies par le groupe, pour la présentation de son travail,
- **homogénéité** : il est bon que les aptitudes techniques et intellectuelles des uns et des autres soient de niveau comparable et si possible complémentaires.

Cette approche du travail de groupe a été définie par Weinberg en 1971 sous le nom de programmation impersonnelle. Elle s'accommode mieux de l'existence de disparités dans le groupe, ou de déséquilibres dans la pression de travail lors de périodes de crises, qu'un groupe régi par la responsabilité individuelle des sources, où l'ego matérialisé dans les programme se manifeste par le refus de la critique. Elle garantit par ailleurs que les compétences complémentaires de chacun profitent à l'ensemble de la production. Elle n'autorise toutefois toujours pas un membre du groupe à prendre une décision autoritaire sur un programme écrit initialement par un autre sans en tenir ce dernier informé. Dans aucune équipe, fût elle organisée en programmation impersonnelle, on ne peut en général traîner le fardeau d'un individu récalcitrant aux principes élaborés par le groupe, qui nécessite des débats fréquents sur la qualité de sa manière de mettre en oeuvre les décisions.

Dans une équipe de programmation impersonnelle, on accepte qu'il y ait des erreurs à toutes les phases du projet, pourvu qu'on puisse les détecter avant la recette finale. L'équipe est donc également solidairement responsable des tests qu'elle met en oeuvre sur ses programmes. Les avantages de l'approche sont nombreux, mais il en est un absolument saillant : la compétence sur les programmes étant répartie au sein de l'équipe, même si on n'évite jamais que chaque programmeur ait des domaines d'activité privilégiés, l'équipe est protégée contre le départ momentané ou définitif d'un de ses membres. Cela donne plus de souplesse au chef de projet, qui peut sans douleur moduler son groupe en fonction de besoins externes, qui peut envoyer ses développeurs en formation ou à des congrès, qui peut obtenir du groupe une répartition spontanée de la charge sur l'ensemble des membres. Dans le groupe à responsabilité individuelle, le départ d'un membre est souvent critique, et met[7] l'équipe entière en danger si la compétence perdue effraye tout le monde. C'est à tel point vrai que tel chef de projet craindra de se séparer d'un membre néfaste au groupe par son attitude, simplement car il sera perçu comme irremplaçable. Le groupe entier gagne pourtant à se mettre en danger apparent du fait de l'anonymat des programmes (on peut être plus facilement licencié, cela va de soi), car les chances de terminaison du projet sont accrues.

Un autre avantage est un effet de formation permanente. Les compétences de chacun diffusent naturellement dans ses programmes et sont vues et discutées par tous.

---

[7] C'est toujours une apparence cependant.

Les normes définies par le groupe sont mieux acceptées s'il existe des outils pour les mettre en oeuvre et pour détecter les défaillances. Par exemple, la manière dont les programmes sont commentés, et indentés peut être définie et formaté à l'aide de l'outil "indent" sous Unix. Si une telle mise en forme est réalisée à chaque compilation par le "builder" d'exécutables, le programmeur s'habitue vite à voir ses sources selon le format commun, et l'utilise après spontanément, ou au moins le lit sans difficultés. Il va de soi que les normes à définir sont mieux acceptées si elles font l'objet d'un consensus, ou résultent de l'arbitrage d'un chef technique apprécié.

3.5.3. l'influence des conditions matérielles

Le groupe étant un microcosme, il est organisé comme une société et doit favoriser tant la productivité professionnelle de chacun, que ses besoins sociaux. On a souvent observé l'analogie qui existe entre le lieu de travail moderne dans les grandes entreprises, et la rue principale des petits villages, lieu de communications et d'échanges informels nécessaires. L'entreprise doit donc permettre à la fois l'exercice agréable du travail, et des rapports sociaux faciles et qui ne soient pas gênants (pour ceux qui travaillent).

3.5.3.1.espace de travail

L'espace de travail est organisé pour favoriser le calme et les communications productives. On sait que les architectures ouvertes sont impopulaires. En fait, de nombreuses entreprises conservent un espace ouvert ou vitré pour les développeurs, et réservent des bureaux fermés à leurs cadres. L'architecture ouverte est peu productive à cause du bruit de fond des conversations, de la déconcentration qui résulte d'une conversation voisine intelligible, du tabac pour les non fumeurs. De plus, si la place allouée à chacun n'est pas suffisante, la personnalisation du lieu de travail est rendue impossible, et le plaisir de s'y retrouver réduit.

Un espace même semi ouvert doit garantir une intimité suffisante, et permettre une conversation privée entre deux personnes. Un espace même fermé doit permettre à chacun de disposer d'un éclairage naturel, qui est préféré au tout électrique, tout en évitant que les écrans ne fassent l'objet de reflets. C'est la quadrature du cercle à laquelle sont confrontés les architectes.

3.5.3.2.espace de discussion informelle

Le lieu de travail doit fournir un espace clos et confortable permettant la réunion informelle de quatre à cinq personnes autour de commodités (machine à café, frigo), et à portée de la documentation d'entreprise (revues, nouveaux ouvrages achetés), mais loin des imprimantes et des photocopieurs.

4. un point de vue comportemental sur la productivité et la qualité

4.1. introduction

Ce chapitre décrit les résultats obtenus par une étude détaillée, aux états unis, d'un grand nombre de projets informatiques. Il met l'accent sur trois difficultés fondamentales apparaissant dans les grands projets logiciels, à savoir la compréhension du problème par les informaticiens, la variabilité des spécifications, et les problèmes de

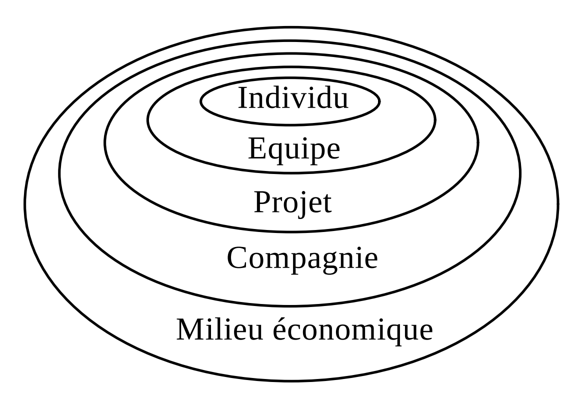
communication. Il éclaire au passage les rôles des individus clef du projet : le programmeur, l'ingénieur intégrateur ("systems engineer"), et le chef de projet.

L'approche est stratifiée, en ce que elle aborde ces aspects à cinq niveaux sociaux distinct : l'individu, l'équipe, le projet, la société et le milieu économique.

Elle est **écologique** au départ, car elle tente de décrire les effets de l'environnement sur l'individu, mais semble démontrer au final le rôle majeur de certaines personnes dans les projets : les gourous techniques. L'analyse qui est faite est donc également **interactionniste**, car montrant les variations de productivité et de qualité associées aux différences entre les individus et entre les situations.

4.2. un modèle stratifié du comportement

Une grande difficulté pour les chercheurs en génie logiciel est de caractériser les conditions par lesquelles une méthode ou un outil peut avoir un impact au niveau des organisations. Pour y parvenir, un modèle hiérarchisé, ou stratifié, des différents acteurs du processus d'élaboration d'un logiciel a été défini : en anglais le "layered behavioral model of software developpement processes", qui est présenté dans [Curtis, Krasner, Iscoe 88]



Dans le modèle stratifié des comportements, l'analyse porte pour les individus sur la motivation et l'aptitude, pour les niveaux équipe et projet sur la dynamique de groupe, et pour les deux autres niveaux sur le comportement des organisations. L'objectif de cette approche est de cerner par une étude précise des communications entre niveaux quels sont les effets de changements ou d'événements survenus à un niveau quelconque sur les individus eux mêmes (il s'agit alors d'une perspective écologique : effets du milieu sur les individus) ou sur les produits des compagnies et le milieu (la perpective est alors économique : quelles sont les causes de pertes et de profit notamment). Dans le cadre du génie logiciel, on se pose souvent la question de l'effet de telle ou telle méthode sur la productivité, ou sur la qualité du produit fini. Le modèle stratifié permet d'aborder l'étude des phénomènes à chaque niveau et leur propagation aux niveaux voisins pour précisément tenter de répondre à ces questions. On a pu constater en effet que l'utilisation de méthodes de génie logiciel permet en moyenne d'accroître la fiabilité des logiciels de 30 % seulement (source Curtis 88), mais n'a aucun effet positif sensible sur la productivité.

Lors d'un projet, l'interaction se fait du milieu économique où se trouvent les clients vers les développeurs, et en retour vers le milieu économique qui accueille le produit. On voit à quel point il est important de savoir à quel point un aspect individuel se propage de façon significative vers le milieu économique.

4.3. la (mé)connaissance du domaine applicatif

Une équipe de programmeurs est compétente dans son domaine professionnel, l'informatique. La réalisation d'applications pour des clients étrangers au monde de l'informatique (c'est le cas général) suppose une compréhension somme toute élevée de la perspective du client, et donc des caractéristiques techniques de ce que l'on appelle le domaine d'application. En fait, le génie logiciel consiste, en amont du processus de développement, à décrire des méthodes par lesquelles un client et un prestataire peuvent s'entendre autour d'un cahier des charges. Comme toute production intellectuelle, un tel document ne peut être compris par les deux parties que s'il est absolument non ambigu. Malheureusement, le savoir implicite du client se manifeste souvent par des "non écrits" qui se révèlent avoir pour le développeur une importance particulière. Plus la connaissance du domaine applicatif est grande au sein de l'équipe technique, plus les chances de voir le projet aboutir, et surtout d'aboutir en ayant été correctement évalué, sont grandes. Cela explique pourquoi les sociétés de service en informatique agissent en général sur des segments très verticaux du marché, pour lesquels elles disposent d'une grande connaissance technique (i.e. ici non informatique), et sont donc placées dans une situation idéale de communication - compréhension mutuelle - avec les clients, et de fiabilité d'évaluation. En plus, fréquemment, elles disposent de chefs de pro jets ayant déjà participé à des projets comparables. Du côté du client, on observe que le choix d'un fournisseur de solution informatique se fait au travers de propositions techniques qui doivent mentionner la réalisation préalable de systèmes comparables, ce qui joue un rôle primordial dans l'acceptation du dossier.

Le cas le plus typique de projets qui nécessitent une connaissance approfondie du domaine applicatif est celui où le système proprement dit est intégré à un ensemble plus large. Par exemple, un programme informatique utilisé dans un réseau de téléphonie. Outre les connaissances techniques nécessaires à garantir la bonne intégration finale du système dans l'ensemble considéré, on doit acquérir des connaissances de détail du domaine qui, bien que non nécessaires à la compréhension de l'esprit du système, s'avèrent cruciales quand il s'agit de programmer. Par exemple, dans le cas du téléphone, on devra peut être tenir compte de la surcharge du réseau qui survient le jour de la fête des mères…

Les chefs de projet confirmés estiment que la difficulté n'est pas de programmer, mais bien de comprendre le problème.

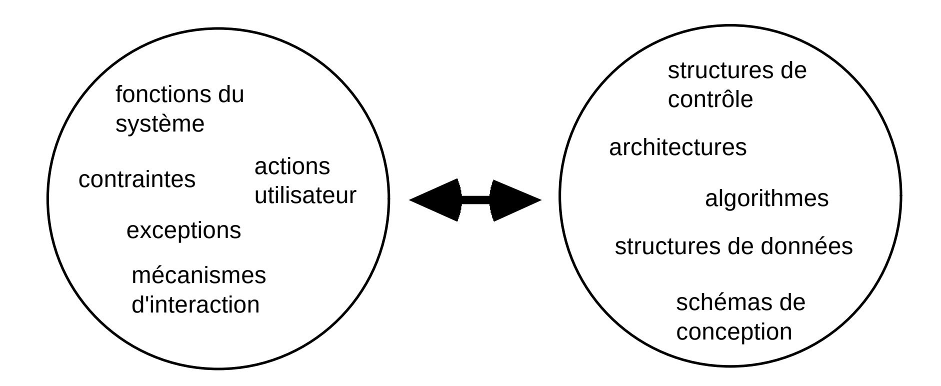
4.3.1. niveau individuel

La motivation, les compétences et l'expérience sont connues comme les facteurs principaux de productivité des individus. Les compétences et l'expérience relèvent soit d'un savoir technique interne à l'activité de programmation, soit de connaissances liées au domaine applicatif. A savoir technique égal, deux individus auront des productivités très hétérogènes si leur connaissance du domaine applicatif diffère. En fait, on peut même considérer comme impossible toute intervention sur un projet dont on ignore tout du domaine applicatif, et donc un développeur qui se trouverait dans cette situation consacre une part importante de son temps en (auto) formation. Le temps consacré à cette tache grève d'autant la productivité.

Un des plus graves dangers dans un projet est de mal interpréter la spécification des besoins. Cela peut se traduire par des erreurs dans la conception (qui se trouve véritablement hors-sujet), erreurs qui lorsqu'elles devront être corrigées se manifesteront par des remaniements probablement importants. De telles erreurs apparaissent quand les développeurs n'ont pas assez d'idées sur le domaine pour comprendre la spécification des besoins. Voici un exemple rapporté par [Leveson 88] : la spécification d'un besoin sûreté dans une installation nucléaire disait : "en présence d'une situation anormale, le système informatique devra laisser toutes les variables de contrôle stables, et faire sonner une alarme". Une panne se produisit, car une vanne trop fermée provoqua une montée de température. Le système informatique réagit conformément à la spécification en laissant la vanne dans sa position, ce qui aggrava l'incident.

L'ambiguïté du besoin tel qu'il est exprimé est la suivante : la variable de contrôle est elle la température ou bien l'angle d'ouverture de la vanne ?

On constate que les individus reconnaissent leurs limites et se subordonnent spontanément aux éléments d'un groupe qui possèdent la compétence la plus vaste du domaine applicatif. Ces personnes sont souvent qualifiées d'exceptionnelles par leurs pairs, sont appelées des gourous (en anglais "project gurus"), et sont fréquemment réputées "avoir sauvé le projet". Elles sont impliquées de façon considérable dans le processus de conception, car elles ont la capacité de projeter le besoin sur une implémentation :



Le concepteur doué est habile à décrire les interactions des divers composants du système, et a parfois défini des notations pour les représenter. Il sait aussi pressentir les fonctions, contraintes et situations exceptionnelles non explicitement spécifiées[8].

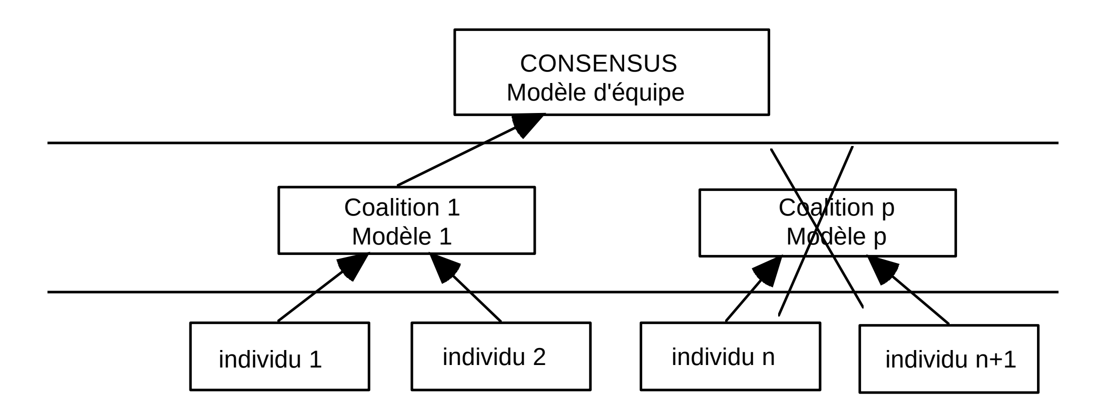
Le coût du développement des grands systèmes intègre implicitement celui de la formation des concepteurs doués. Notons que la compétence d'un concepteur doué se nourrit d'elle même car sa fonction le place au centre de très importants flux d'informations, et le maintient au contact des préoccupations du client, ce qui n'est pas le cas en général pour les développeurs de base.

Des concepteurs doués peuvent tirer une équipe sans quasiment jamais programmer, simplement en orientant l'activité de chacun, et parfois, la conception se fait majoritairement de façon informelle, par le biais de conversations avec les programmeurs.

4.3.2. équipe

La connaissance du domaine joue un rôle au niveau de l'équipe, notamment car elle permet de conforter la position dominante - i.e. le succès en négociation - de certains individus. La phase de négociation principale est celle des choix de conception. Pendant cette période, les avis des uns et des autres se composent jusqu'à atteindre une situation de consensus, qui garantisse à la fois que tous les individus partagent la connaissance du domaine minimale pour comprendre le bien fondé des choix effectués, et que chacun est prêt à s'impliquer dan l'approche définie par l'équipe.

[8] il s'agit des non dits évoqués plus haut, qui résultent de la croyance chez le client que le pretataire connait le domaine applicatif aussi bien que lui.

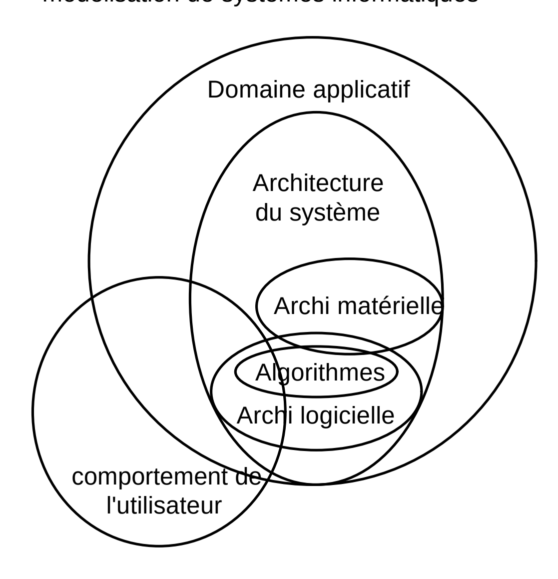


La coalition dominante a en général plus de savoir sur le domaine d'application. Dans des groupes formés de membres d'une seule compagnie, on observe que peu de coalitions rivales se forment, souvent du fait de l'aura d'un concepteur doué. Le débat ne porte alors que sur des alternatives à un modèle de base déjà accepté.

Dans un groupe où la connaissance générale du domaine est faible, on aura souvent autant d'avis que de personnes. La position choisie par un seul individu peut alors faire basculer la décision, notamment quand deux personnes groupées ont une majorité (relative bien sûr). Cela peut donc être une stratégie suivie par deux personnes de se rallier en alternance à l'avis de l'autre, ce qui a pour effet de provoquer des choix rapides, la majorité relative dégagée ayant souvent un fort effet d'entraînement sur le reste de l'équipe. Ainsi, deux concepteurs alliés peuvent tirer une équipe et la faire avancer rapidement dans l'effort de conception, lorsque la perception du domaine génère trop d'ambiguïtés et provoque des débats sans fin.

4.3.3. projet

Il est fait nécessité de garantir que les différentes équipes de conception et de développement partagent une vue unique de l'intégration du système. Cette intégration nécessite des connaissances qui peuvent être décrites à l'aide du schéma suivant.



On voit que les différents aspects ne sont pas indépendants, et que la connaissance la plus large de différents sous domaines techniques donne à celui qui la possède une vision la plus précise des possibilités d'intégration. Dans un projet, l'individu qualifié plus haut de concepteur doué se situe en général au niveau de l'architecture du système. En d'autres termes, il connaît du domaine applicatif tout ce qu'il est nécessaire pour être capable de concevoir un modèle informatique du monde réel.

Bien entendu, maîtriser le projet nécessite de garantir que l'activité des différentes équipes est bien conforme à l'esprit (le domaine technique) et à la lettre (les choix faits lors de la spécification) et notamment n'interprète par l'expression des besoins d'une façon contradictoire. Lorsqu'une dérive est constatée, il faut recentrer l'activité de l'équipe concernée. Cela peut être fait soit de façon consensuelle, ce qui implique des qualités de pédagogie et de négociation, soit de façon brutale. Certains grands projets mettent en place une équipe de "gardiens du temple", une équipe de 5, 6 personnes qui effectue des opérations coup de poing. Lorsqu'une dérive est constatée, ils interviennent directement sur la conception et les programmes en un temps très court, pour restaurer la conformité. Les auteurs se sentent agressés mais les effets sont immédiats, et la ligne commune est bien mieux suivie par la suite.

4.3.4. compagnie

Au niveau de la société, la question de la connaissance du domaine applicatif et du savoir technique se pose pour les managers. Quand ils sont issus de la programmation, leur compétence subit rapidement l'usure due à l'émergence des techniques nouvelles et à l'érosion de leur savoir ancien. Le maintien de compétence se fait essentiellement par la communication, et on observe souvent que le contact avec l'évolution des techniques est maintenu grâce aux ingénieurs qui reviennent de formations ou de congrès et négocient avec le manager l'achat d'outils nouveaux.

Au plan stratégique, les managers sont responsables de l'occupation des niches de marché sur lesquelles la compagnie est la plus compétente et la plus productive. Ils doivent donc veiller à ce que le développement de lignes de produit nouvelles ne soient pas génératrices d'incompatibilités avec l'activité courante. Ils doivent aussi encourager les équipes marketing à définir un discours qui rapporte des affaires les plus cohérentes avec la compétence de la société.

D'un point de vue interne, la plus grande difficulté qu'affronte un manager est d'évaluer les limites de leurs équipes, aussi bien que la pertinence de leurs évaluations de coût et de temps. A ce niveau aussi, une connaissance aussi riche que possible des conditions posées par le domaine d'application sera extrêmement profitable.

4.3.5. milieu économique

Les grands projets sont parfois attribués à des consortiums de circonstance, combinant les compétences et savoirs faire plusieurs compagnies. Dans un tel groupement, chaque société développe ou a déjà développé son propre modèle du domaine applicatif. Cela rend la coopération entre différentes sociétés difficile, tant que les bases d'un compréhension non ambiguë n'ont pas été posées. Dans la mesure ou le choix de faire réaliser une solution informatique par un combinat de sociétés relève parfois de raisons politiques, on doit mentionner l'impact contre-productif considérable d'une telle approche, lorsque l'un des membres du consortium possède de façon interne un savoir faire susceptible de couvrir l'ensemble du domaine.

4.3.6. conclusion

Tout cela montre l'importance d'une gestion de la formation continue, spécialement celle relative au domaine applicatif, qui est un facteur clé de productivité, de qualité et de réduction des coûts.

4.4. la fluctuation et l'incohérencedes exigences

Rappelons que les exigences correspondent au contenu du cahier des charges fourni par le client, puis élaboré de concert avec le prestataire jusqu'à un accord sur les caractéristiques du produit informatique à réaliser. L'expression des besoins est donc un processus très en amont dans le projet, dont les implications dans le produit informatique lui même peuvent être très diffuses, et dont la modification entraîne toujours des remaniements en profondeur, extrêmement coûteux. Les changements de besoins exprimés sont donc une source de contre productivité considérable. Voici les principales sources de fluctuation des besoins exprimés :

4.4.1.1.modifications issues du milieu économique

- clients : nouveaux besoins, modifications de besoins, ajouts
- technologie : progrès des techniques, produits des concurrents
- normes : nouvelles normes, directives d'état

4.4.1.2.modifications générées de façon interne par les sociétés

- technologie : résultats de R et D, compatibilité avec d'autres lignes de produits
- autres : sollicitations des services financiers, juridiques, du marketing.

4.4.1.3.changements dus aux individus

- il s'agit ici des effets pervers des développements spontanés, correspondant à des extensions des contraintes initiales non demandées par le client. Ces initiatives individuelles sont cachées par les programmeurs eux mêmes, ou par le chef de projet, n'apparaissent jamais dans les documents d'expression des besoins, mais pèsent sur ces derniers.

Du fait de l'importance dominante du milieu dans la fluctuation des besoins, par le fait du client lui même notamment, la présentation suivante débute par cet acteur principal.

4.4.2. milieu économique

Il est fréquent qu'une société bâtisse une activité sur la base d'un premier client. Dans ce cas, les besoins pris en compte pour la spécification du produit sont ceux exprimés par ce premier client. L'élargissement du marché à de nouveau clients se fait en incorporant les besoins issus de clients successifs, dans le pire des cas. La fluctuation des besoins est alors acceptée par la société comme un mode de travail normal, et les incohérences qui peuvent en découler acceptées d'avance. Cette méthode ne remplace jamais une véritable vision stratégique, ni le bénéfice d'études de marché. Notons qu'un des inconvénients majeurs de l'approche provient des relations trop intimes entretenues avec le client initial, qui se sent en position de force pour exiger des adaptations du produit qui peuvent être néfastes, car ne résultant pas d'une bonne analyse de ses besoins propres, et notamment contradictoires avec le reste du marché, ou avec une politique correcte de développement.

La dialectique définie par un prestataire pour expliquer son projet, qui est le résultat de méthodologies rigoureuses d'analyse, donne parfois de nouvelles idées au client en améliorant à la fois sa propre compréhension du besoin, et en faisant miroiter des possibilités non encore envisagées, qui sautent aux yeux dans une présentation çà l'architecture élégante. Le client regrette alors de ne pas avoir intégré ces fonctions à l'expression des besoins et n'aura de cesse que de les voir implémenter, gratuitement si possible, par un patient travail de sape et de négociation. La rigueur de gestion d'un projet dépend donc souvent de la fermeté de l'attitude d'une compagnie par rapport à ses clients, notamment en matière d'évolution des besoins. Par ailleurs, il existe un principe intransgressible : on ne doit jamais faire valoir sa compétence en laissant entrevoir la possibilité de réalisations ambitieuses. On ne doit vendre que ce que l'on a, jamais ce que l'on n'a pas encore.

L'accord entre client et prestataire se fait parfois après de nombreux échanges de documents, cahiers des charges dans un sens, propositions techniques dans l'autre, jugées alternativement trop chères ou trop incomplètes par le client, jusqu'à satisfaction des deux parties. Lorsqu'une telle phase intervient , il va de soi que les besoins exprimés sont modifiés souvent, et que le développement ne peut ni ne doit commencer.

Notons enfin qu'il est particulièrement nuisible au concepteur de laisser un accès direct des clients à l'équipe technique. Le client qui a accès à l'équipe de développement fait passer des besoins sous le manteau, si possible gratuitement, ce qui est possible car les gestionnaires ne peuvent en tenir compte. Malheureusement, il est parfois difficile de l'empêcher, surtout si le produit est réalisé pour un premier client, qui teste des versions insuffisamment achevées. Le besoin de fréquemment atteindre les programmeurs pour corriger ou contourner des bugs permet d'établir un dialogue favorable à des interventions dangereuses du client.

4.4.3. compagnie

De façon interne, une compagnie peut générer des fluctuations de besoins sur un projet. Le cas est flagrant dans les sociétés éditrices de produits logiciel, au sein desquelles le service marketing se substitue souvent au marché pour imposer une vision des fonctionnalités du système à réaliser (d'ailleurs pas toujours pertinente). Fréquemment, la perception des besoins, et la vision du produit varient entre le groupe marketing, le groupe décision stratégiques, et le groupe organisation des produits au sein d'une même société.

Le service marketing a souvent l'impression trompeuse que le produit qui sera vendu le plus facilement est celui qui fait tout. Dans un tel contexte, l'équipe de conception atténue les contradictions qui apparaissent en essayant de décrire un produit cohérent, et restituant au mieux la vision des uns et des autres. C'est à ce niveau que des interventions autoritaires d'un manager fondées non pas sur une claire compréhension du marché mais sur des préconçus commerciaux ou techniques peut être catastrophique. Un besoin nouveau imposé aux concepteurs peut avoir le double effet de forcer un remaniement inutile d'une version déjà aboutie ou presque du système, et de démotiver l'équipe s'il n'est pas justifié. On doit citer cette maxime d'un manager : "Même si la qualité et la performance doivent en souffrir, il est préférable que des gens utilisent vos produits et s'en plaignent plutôt qu'ils n'utilisent les produits de la concurrence."

4.4.4. projet

Un projet peut avoir été décrit en premier lieu pour obtenir un marché. Plus tard, on réalise que certaines contraintes sont incompatibles. Egalement, un projet peut devoir être adapté du fait de l'émergence d'une technologie qui le rend, en tout ou partie, obsolète. De nouvelles contraintes sont alors posées, qui simulent une phase d'améliorations après livraison, avant même que le projet ne soit terminé. Parfois, il arrive que les contraintes soient incompatibles avec les caractéristiques du matériel choisi. Cela impose de revenir en arrière sur un certain nombre de besoins exprimés.

4.4.5. équipe

Pour pallier à des inconsistances entre les contraintes posées par la société et le marché, les équipes mettent parfois en place des structures assez souples (flexibles) pour pouvoir retarder une décision.

Egalement, pour gérer des contraintes incompatibles, il est parfois choisi de les ordonner par priorités, et d'en inclure autant que possible selon cet ordre (efficace quand les conflits apparaissent du fait de limitations de l'espace mémoire).

Même lorsque les contraintes sont stables, des fluctuations des spécifications peuvent apparaître du fait d'une mauvaise coordination des développements des différents composants.

Le ratio de composants du système n'ayant pas encore fait l'objet d'une modélisation informatique (conception) par rapport au nombre global donne une bonne mesure du risque rémanent de dérive des spécifications d'un système. Il arrive que des inconsistances ne soient détectées qu'à partir de l'intégration.

4.4.6. niveau individuel

De nombreuses contraintes apparaissent lorsque l'implémentation force à poser certaines questions techniques précises. Les programmeurs sont également souvent confrontés à des ambiguïtés de l'expression fonctionnelle des besoins.

Une source de modification des contraintes est due aux programmeurs qui prennent la liberté d'étendre et abstraire les fonctions d'un système (pour des raisons d'élégance souvent). Un manager français d'une société qui réalise des produits technologiquement avancés déclare : “ Ici, la seule recherche qui est faite est la recherche du client.”

4.5. les défauts de communication

4.5.1. niveau individuel

La production de documentation par l'individu est liée à la pression exercée sur lui par les contraintes de bonne fin du projet. Plus le projet avance, plus en général la production de documentation est réduite, et donc inconsistante avec l'état du projet. En d'autres termes, on écrit en début de projet une documentation qui est correcte pour un état temporaire, et on ne peut plus la modifier par la suite. L'inconsistance de la documentation est identifiée par les individus comme une des principales sources de difficulté de communication dans les grands projets. En effet, elle accroît les besoins d'accès direct à l'individu, et est un facteur de dérangement. C'est une des raisons pour laquelle Bertrand Meyer considère que plus la documentation peut être extraite des programmes eux mêmes, meilleur c'est, car cette documentation là est consistante et à jour.

Les réseaux individuels de communication sont liés aux questions principales suivantes :

- fonctionnalité : que faire?
- connaissance technologique : quels outils?
- gestion : quel est le risque?
- diagnostic d'erreur : pourquoi ça ne marche pas?

- architecture système : quelle est l'interface?
- composants disponibles : que puis je réutiliser ?

Chaque individu possède un certain nombre d'interlocuteurs connus pour chaque type de question, qui constitue son propre réseau de communication. Dans un groupe où existe un gourou, on voit que l'ensemble des réseaux peut tourner autour de cette même personne, ce qui constitue une raison suffisante de constituer des groupes où la connaissance technique est homogène.

4.5.2. équipe

Dans les équipes ayant conclu un projet avec succès, on observe souvent que les ingénieurs d'intégration ont défini des conventions de représentation communes à l'équipe, acceptées et utilisées par tous. Ces conventions sont parfois graphiques, leurs auteurs sont inflexibles quand aux notations. Une notation dépourvue d'ambiguïté en ce qui concerne le modèle du système semble donc être garante d'une diffusion de la compréhension du système, et d'une bonne communication au sein de l'équipe.

On note également que dans les phases préliminaires du projet, les conventions de nommage sont discutées autant que la décomposition du système.

4.5.3. projet

Il est difficile pour un chef de projet d'établir une communication avec ses équipes quand cette relation ne s'établit pas spontanément. Le chef de projet éprouve donc en général de la difficulté à obtenir une connaissance du système effectivement mis en oeuvre. Puisqu'il est responsable de la gestion du projet, il met en place des revues régulières du projet, et on observe que ces points de rencontre avec les équipes servent justement plus à communiquer qu'à détecter d'éventuelles erreurs. Les équipes pratiquent d'ailleurs autant que possible des tests avant les revues et connaissent bien la liste des défauts du système en l'état.

La communication est souvent améliorée au niveau du projet quand une personne établit le lien entre différentes équipes. Certains projets attribuent ce rôle d'office à un ingénieur d'intégration en chef ("chief systems engineer"). Cela permet parfois de maintenir le degré de communication minimal requis entre des équipes rivales.

Notons aussi que la communication s'établit au sein du projet en fonction des liens logiques du système, mais également en fonction de liens sociaux. Un projet réussi organise les équipes de manière à respecter certaines réalités.

4.5.4. compagnie

établissement de processus formels de revue lors de la réalisation de grands systèmes. Inefficaces pour l'identification de difficultés apparaissant dans des domaines non couverts par le modèle. Des contacts personnels informels s'avèrent la meilleure méthode le meilleur moyen de communiquer de l'information au travers des frontières administratives.

Quand des groupes aussi divers que le marketing, l'ingénierie des systèmes, le développement, l'assurance qualité, ou la maintenance expriment leurs besoins, ils montrent souvent des lacunes dans l'échange d'information[9]

Même lorsque toutes les équipes mises en oeuvre sur un projet appartiennent à la même division, des faiblesses dans la communication naissent de la simple succession des phases, qui nécessite un transfert de compétence à chaque stade. Voici un exemple des équipes qui peuvent intervenir sur un projet :

[9]parfois, les projets réalisés pour la défense favorisent cette situation car ils reposent sur des informations confidentielles.

- équipe de gestion de projet
- équipe de spécification des besoins planification, expression des besoins
- équipe de spécification détaillée analyse des besoins, définition des fonctions
- équipe de développement conception, réalisation, tests
- équipe de livraison vérification, installation, formation
- équipe de maintenance spécifique, évolutions, correction de bugs

Bien que la discussion soit préférée à la documentation, cette dernière apparaît comme la principale source d'échanges entre les équipes successives.

4.5.5. milieu économique

Il y a une nécessité fréquente de recourir à une communication direct entre développeur et client, pour éviter les intermédiaires. Pourtant, l'organisation administrative de la communication dans l'entreprises le rend en général impossible. Ce qui est requis, c'est un point de contact chez le client permettant de clarifier les spécifications.

Dans les projets officiels (militaires) le prestataire est souvent confronté à plusieurs entités administratives qui prétendent toutes à des titres divers être "le client" : le ministère, le fabricant d'hélicoptères, la société ayant obtenu le marché (qui sous traite souvent la réalisation), les militaires eux mêmes qui utiliseront le système. Cela ne facilite pas la communication, ni la définition précise des besoins.

Parfois les cycles d'approbation de modification du cahier des charges sont si longs que des modifications nécessaires ne sont pas réalisées.

Les clients disposent souvent de scénarios de mise en oeuvre du système, qui sont trop rarement passés aux équipes de spécification et développement, auxquelles ils seraient extrêmement utiles.

4.6. Conclusion

Il s'avère que le développement de grands systèmes informatiques doit être vu comme un processus d'apprentissage, de négociation, et de communication.

4.6.1. conséquences sur les outils et la pratique informatique

Les concepteurs d'environnements de développement doivent découvrir des structures permettant de favoriser de façon créative le partage et l'intégration de connaissance par l'ensemble des individus impliqués dans un projet.

Les outils et techniques doivent considérer le changement comme un processus naturel, et permettre la représentation de décisions de conception incertaines.

Les qualités qu'un outil doit apporter sont donc le partage et l'intégration de connaissance, la prise en charge de l'évolutif, et des facilités de communication et de coordination.

Si une de ces qualités manque à un outil, il semble peu probable qu'il puisse avoir un réel impact sur la qualité et la productivité.

4.7. conséquences sur la gestion de projet

On observe dans tous les projets l'importance fondamentale du facteur humain, qu'il s'exprime au travers de seuls concepteurs doués, ou qu'il soit réparti sur l'ensemble des équipes. Brooks déclare : "la question de savoir comment améliorer l'art du logiciel repose, comme il l'a toujours fait, sur les individus". Et Boehm "les qualités individuelles et les activités relationnelles constituent de loin la plus importante source de gains de productivité et de qualité". On trouvera aussi des managers pour déclarer : "la seule règle que j'ai dans ma gestion est de garantir que je dispose d'individus brillants - réellement brillants -, que je forme des individus brillants, et que procure un environnement où des individus brillants peuvent produire".

Il n'est pas surprenant que les individus jouent un rôle central vu la nature et la complexité des compétences à mettre en oeuvre dans un grand projet informatique, qu'aucun outil n'est en mesure d'automatiser. Cela dépasse la simple aptitude à programmer, et couvre des domaines comme : la résolution de conflits de besoins, la négociation avec le client, la garanti que l'ensemble des équipes partagent bien une vision identique du système, la possibilité de maintenir la communication entre des groupes distincts et concurrents.

4.8. conséquences sur les modèles de processus logiciels

Les modèles habituels se focalisent sur les étapes habituelles du développement : spécification des besoins, spécification détaillée, conception, développement, etc.… On observe dans cette étude qu'il faut y ajouter de façon transversale des processus cruciaux comme :

- la formation interne
- la communication technique
- la négociation des besoins
- la communication avec le client

Les modèles habituels organisés sur la base d'une succession de taches n'aident pas à analyser les besoins de l'équipe de développement en formation, comment négocier avec le client lors d' inconsistances de besoins, comment les équipes résolvent les conflits architecturaux, et comment ces facteurs, et d'autres similaires, contribuent au risque.

Quand on superpose à la vision temporelle du processus une vision comportementale intégrant les aspects cognitifs, sociaux et organisationnels, on a une meilleure chance de percevoir les sources de contre productivité et de manque de qualité. Plus un chef de projet comprend profondément ces processus comportementaux, plus il a une vision précise des facteurs qui contribuent à la réussite.

5. la maintenance

On considère que la maintenance est à peu près pour 15 % corrective (bugs d'implantation, de conception, de spécification), pour 20 % évolutive (évolutions programmées ou devenues nécessaires du produit), et pour 65% de mise au point. Elle représente en tout cas à peu près la moitié de l'effort total sur un produit.

La maintenance demande les mêmes procédures de suivi que le développement (de gestion, d'assurance qualité notamment). Elle est confrontée au problème de la **régression**[10], et doit donc avoir accès à l'ensemble des documents de spécification.

Les équipes de maintenance et de développement possèdent des statuts en général similaires, et travaillent de façon totalement indépendantes. Lorsqu'un produit devient commercialisable, ou livrable[11], car avalisé par l'assurance qualité, il est transféré des équipes de développement vers celles responsables de la maintenance.

5.1. la dynamique d'évolution des programmes

Des observations statistiques ont conduit aux lois de Lehmann :

- évolutions : un programme est de moins en moins adapté à son environnement d'utilisation, et doit donc évoluer

- complexité : la croissance de la complexité est croissante au fil des évolutions d'un programme (on doit donc tout jeter à intervalles réguliers)

- versions : les intervalles de temps entre versions, comme les évolutions de la taille d'un produit semblent constantes, de même que les évolutions fonctionnelles apportées à ce produit

- productivité : l'effort de développement (taille des équipes) semble n'avoir aucun effet sur les volumes produits (la productivité décroît au rythme où croissent les effectifs.

5.2. les coûts de maintenance

Les coûts de maintenance sont supérieurs aux coûts de développement, et leur sous estimation est dramatique. Par exemple, un produit livré à des premiers clients avant d'être complètement terminé, et dont la maintenance est assurée par l'équipe de développement principale[12], génère des coûts de maintenance qui peuvent noyer l'équipe, retardant "ad vitam aeternam" les échéances à venir.

Il est donc rentable d'investir dès la conception sur la réduction de ces coûts, au moyen d'actions organisationnelles et de contrôle qualité. On peut agir ainsi sur :

- des facteurs non techniques : stabilité de l'équipe, durée de vie du programme, dépendance du programme vis à vis de l'extérieur, stabilité du matériel

- des facteurs techniques : indépendance des modules, langage(s)[13] de programmation, style de programmation[14], méthodes et outils de validation et de test, documentation, gestion de configuration

---

[10] il s'agit du phénomène par lequel une modification, ou une correction engendre un défaut dans une partie du programme qui en était exempte

[11] en anglais : “ready for shipping”

[12] cette situation très mauvaise est généralement observée dans les petites "start up" informatiques".

[13] plusieurs en général

[14] en anglais : “good programming practices”

5.3. l'obligation de maintenance

Un produit vendu fait l'objet d'une garantie par son réalisateur, en général facturée au client sous la forme de contrats de maintenance intéressants car générateur d'un chiffre d'affaires permettant le financement de cette activité. Le chef de projet doit dans tous les cas contrôler les coûts induits par les interventions de maintenance demandées par les clients. En particulier, toute demande de maintenance ne se traduit pas par une demande interne de modification. On considère deux types de bugs :

- le bug bloquant : par lequel un client est incapable d'utiliser l'outil comme prévu, ce qui l'empêche de travailler (par exemple : un produit de gestion dont un pan de la facturation ne fonctionne pas correctement). Ce bug doit être corrigé immédiatement par la maintenance, la version corrigée envoyée au client, et un avis de bug[15] transmis aux équipes de développement.

- le bug non bloquant : la maintenance est capable d'indiquer au client une procédure réaliste ("patch") qui permet de contourner le bug, ou de supprimer ses conditions d'apparition (création d'un client fictif par exemple dans un système de gestion ne fonctionnant pas avec zéro client). Ce bug ne doit être pas nécessairement corrigé, le client attendant alors la livraison d'une prochaine version mineure.

Notons que dans ce deuxième cas, si le défaut n'apparaît que chez une faible proportion de clients, et si le coût de modification est trop élevé, on peut décider de ne pas le corriger. Il peut alors demeurer jusqu'à la fin du produit, ou disparaître spontanément si une prochaine version majeure possède une conception qui supprime ses conditions d'apparition.

Les demandes d'évolution transmises par les clients sont encore moins prioritaires.

6. la gestion de configuration

6.1. le plan de configuration

Le chef de projet définit les règles de nommage des fichiers, les liens entre documents formels, désigne un responsable de la vérification de ces documents, un responsable de leur livraison à la gestion de configuration. Egalement, il procède :

- à la définition des répertoires de travail

- au nommage des responsables qualité

- à la définition de la bd de configuration

- à la détermination des procédures de contrôle des modifications, de construction du système, de gestion des versions

[15] en anglais : “bug report”

On doit considérer au minimum 6 documents par programme: la spécification (des besoins), la (spécification de) conception, les sources, les spécifications de tests, le manuel utilisateur, le manuel de référence. Il est nécessaire par ailleurs de gérer de très nombreux fichiers (objets, exécutables, include, lib, etc...). On doit souvent recourir à une base de données pour archiver toutes les données de gestion du système : clients, configurations, constitution de version etc...

6.2. le contrôle des modifications

Les modifications doivent faire l'objet d'une gestion rigoureuse, quelles que soit leur provenance. Les étapes à suivre sont alors :

- enregistrement d'une demande de modification
- analyse d'impacts, analyse de risque, prise de décision ou rejet de la demande
- réalisation de la modification
- revue de l'assurance qualité (respect des standards, régression)
- livraison

6.3. la construction du système

Un produit est toujours obtenu au travers de passes parfois nombreuses de compilateurs et de formateurs. L'outil le plus utilisé est "**make**" sous Unix, dont le fonctionnement s'apparente un peut à un système expert à règles, qui décide de recalculer un exécutable en fonction des dates de dernières modifications des fichiers. Make peut être vu comme un gestionnaire de l'intégrité des objets par rapport à leurs sources. La bibliothèque graphique X/Window a introduit un outil plus général appelé **imake**, qui génère automatiquement des fichiers de règles "makefile" pour make, garantissant ainsi la portabilité des makefiles. De nombreux ateliers de génie logiciel incorporent un outil appelé "builder", qui permet de passer outre certaines restrictions de "make", et ainsi de travailler en parfaite conformité avec les décisions prises pour la gestion de configuration (dont la conception peut ainsi s'affranchir de la considération des mécanismes de make).

6.4. la gestion des versions et des évolutions

Un mode simple et assez répandu de suivi des différentes versions d'un produit s'appuie sur une numérotation à trois chiffres. Par exemple : produit v1.7.2. Dans cet exemple, la signification des différents numéros est la suivante :

- le 1 : c'est le numéro des versions majeures du produit, dont la sortie s'accompagne de progrès importants au niveau des fonctionnalités, et/ou d'un changement notable d'environnement d'utilisation, ou de portabilité.

- le 7 : c'est celui des versions mineures. L'incrément est réalisé à chaque fois que l'équipe de développement libère une version du produit qui corrige des bugs attendus par les clients (mais non bloquants), et apporte des modifications légères.

- le 2 : c'est le numéro des corrections : versions résultant de la maintenance.

Habituellement, les versions majeures peuvent être délivrées tous les 18 mois à deux ans, les version mineures tous les six mois, et les versions mineures à un rythme aussi rapide que nécessaire.

On utilise également les dénominations alpha, beta de la manière suivante : la v1.7 à peine terminée par les équipes de développement s'appelle v1.7.alpha durant la phase des tests et des revues de qualité chargées de déterminer sa conformité. Celle ci acquise, elle est dénommée v1.7.beta, donnée en charge à l'équipe de maintenance, et en général livrée à un panel de clients privilégiés appelés beta sites, pendant une période probatoire de deux à six mois. A l'issue, la livraison du produit en masse est possible, à partir de la version v1.7.0. Bien sûr, il s'agit ici plus d'un exemple de ce qui est possible que d'une règle.

6.5. les outils de gestion de configuration

Les outils de gestion de configuration permettent la mise en oeuvre des standards définis par le plan de configuration. En général, sous Unix, ces outils sont les répertoires, l'outil de contrôle de code source "sccs", et make lui même qui permet de nommer et transporter automatiquement des fichiers.

On aura intérêt à concevoir des outils d'extraction des sources pour la constitution des versions. Rappelons en effet que la maintenance opère sur un produit issu du produit développé, qui est perçu comme entièrement nouveau.

6.6. la documentation

Cette partie définit un certain nombre de règles de gestion (classification) et de réalisation (qualité) de la documentation.

6.6.1. la classification des documents

La documentation est partagée entre celle relative au processus de développements (plannings, revues, maintenance …) et celle relative au produit.

6.6.1.1.documentation associée au processus

On a ici des documents qui ne sont pas tous formels, et qui présentent une très grande variabilité. On y trouve notamment les thèmes suivants :

- planning, plans, estimations

- des rapports

- définition de standards

- documents de travail

- le mail échangé par les différents intervenants

6.6.1.2.documentation associée au développement

6.6.1.2.1.documentation fournie à l'utilisateur du système

On a notamment : le manuel d'installation, le manuel d'administration, le manuel d'utilisation, le manuel de référence

6.6.1.2.2.documentation liée au système

On y trouve : le cahier des charges, l'architecture du système, l'archivage des programmes et des listings, les documents de validation, le guide de maintenance

6.6.2. la qualité des documents

6.6.2.1.la structure d'un document

Un document comporte nécessairement une page d'en tête :

- qui le situe dans le projet (référence de projet, auteurs, catégorie du document)
- qui décrive sommairement son contenu (titre parlant, résumé)
- qui le date et donne sa version
- qui en permette l'archivage (mot clef, type de document)
- qui décrive les standards auquel le document est supposé se conformer
- qui en décrive le copyright
- qui en décrive la circulation souhaitée (nominatif et/ou classement de confidentialité)
- qui donne un contact pour questions ou remarques
- qui dise s'il s'agit d'une version préliminaire[16] (de travail) ou définitive

Ces différents aspects sont indicatifs d'une liste peut être plus générale, suivant l'organisation concernée.

6.6.2.2.les standards de documentation

6.6.2.2.1.les standards en matière de processus

Ces standards décrivent l'approche à suivre pour la réalisation de documents, sous la forme notamment d'outils à utiliser, de normes de présentation, et de procédures d'assurance qualité. On considère en général la vie du document comme étant constituée de trois étapes : la création d'un brut de travail permettant une revue sur le contenu du document, puis une phase dite de "pré-publication" de corrections de forme (orthographe et grammaire doivent être

[16]“draft” en anglais

parfaites en sortie), et enfin une phase de publication plus ou moins artistique, qui sera effectuée par des personnels non techniciens.

Notons que si un effort permanent de forme pendant la phase un n'est pas souhaitable (il peut être inutile si des parties du texte sont supprimées avant la pré-publication), il n'est jamais bon de faire lire un document truffé de fautes.

6.6.2.2.2.les standards en matière de produit

La documentation doit faire l'objet de définitions précises. Parmi les différents standards à définir pour un projet, ou une organisation, on peut mentionner :

- les normes d'identification des documents
- les contraintes de structuration (organisation de documents de chaque type)
- les guides de **style** et règles de *présentation*
- les modalités de mise à jour (utilisation de barres de marge pour mettre en évidence une partie nouvelle par rapport à une version antérieure de document)

6.6.2.2.3.les standards d'interchangeabilité

Les contraintes de communication au sein d'un projet peuvent influer sur les conditions techniques de réalisation de la documentation. Par exemple, si des fichiers doivent pouvoir être exploités indifféremment sur des environnements très différents, avec des outils distincts, ou de versions différentes, il faudra peut être se limiter à un niveau assez réduit de qualité d'aspect. Sans aller jusqu'à définir un protocole edi (échange de données informatisé), il peut être utile de travailler en mode texte formaté à l'aide d'outils comme **Latex**, ou bien sur la base d'un standard de description de documents normalisé comme l'est **sgml**. Par exemple, on pourra décider l'utilisation généralisée du langage de description d'hypertextes de Mosaïc, si l'on veut publier la documentation au travers d'un serveur de fichiers w3.

6.6.2.3.le style rédactionnel

Bien qu'il ne s'agisse pas d'une considération rédactionnelle, mais relative au processus individuel de réalisation de la documentation, disons ici qu'il est préférable d'aborder un document par sa structure (le plan) que par le texte. Certains logiciels de traitement de texte (Word) permettent d'ailleurs de travailler en mode "plan". Egalement, on conseille d'écrire l'introduction en dernier. Voici une liste de préceptes à considérer lorsqu'on rédige de la documentation :

- séparer clairement les paragraphes qui peuvent être perçus comme des réponses aux questions quoi?, comment?, pourquoi?, par qui?, ou?
- identifier des niveaux de texte correspondant à des lectures plus ou moins détaillées. On envisage habituellement trois niveaux : les titres, le corps principal de texte, et enfin le texte qui peut être réservé pour une seconde lecture (qui peut être renvoyé en annexe)
- mentionner en notes de bas de page les considérations à caractère anecdotique qui même si elles éclairent le sujet perturbent la compréhension d'une phrase

- mettre en évidence la première apparition d'un terme dans le texte, et surtout une mention qui le définit, par exemple en utilisant des caractères gras pour le mot nouvellement défini. Une définition ne doit pas pouvoir échapper à l'attention, même lors d'une lecture rapide
- écrire des phrases et des paragraphes courts
- ne pas utiliser de double négation
- utiliser des formes verbales actives, impératives, et le présent
- avoir une bonne orthographe (imposer l'usage d'un correcteur), et une bonne grammaire
- définir les termes utilisés : un glossaire doit impérativement accompagner tout document
- se répéter si nécessaire lorsqu'un concept est difficile. Multiplier les points de vue peut permettre à un lecteur de trouver un point d'ancrage sur le sujet
- donner des références explicites vers des documents annexes ou bibliographiques, en rappelant le contexte cité.

7. l'assurance qualité

On peut définir brièvement les trois notions centrales de l'assurance qualité de la façon suivante :

- la **qualité** est le niveau atteint par un produit pour certaines de ses caractéristiques et de ses fonctions. Comme on dispose généralement de nombreuses fonctions, il est rarement possible de dire si un produit est de qualité meilleure qu'un autre[17]. En ce sens, la notion de qualité est subjective.
- l'**assurance qualité** est un processus par lequel on évalue la conformité d'un produit à sa description préalable (vérification) et aux besoins du client (validation).
- le **contrôle qualité** est l'ensemble des techniques et outils mis en oeuvre pour accroître la qualité du produit réalisé[18]. Le contrôle qualité permet d'agir sur certains des critères possibles de qualité, mais pas tous.

Il est important de noter que l'assurance qualité est un processus de gestion, qui englobe les notions de vérification et de validation, de nature technique, mais ne s'y limite pas. L'assurance qualité organise notamment la vérification et la validation, et ce tout au long du cycle de vie d'un produit. Pour ce faire, et afin de réduire les coûts inhérents à sa mise en oeuvre, elle peut imposer des contraintes non techniques, comme la lisibilité, la maintenabilité, qui ne sont du ressort ni de la vérification, ni de la validation.

---

[17]l'analyse de la valeur (AV) impose de donner une valeur à chacune des fonctions d'un produit, et donne de ce fait une mesure absolue de la qualité. En comparant valeur totale et coût total, on peut ainsi évaluer une rapport qualité/prix, et choisir en fonction de ce critère objectif.

[18]Une assurance qualité rigoureuse sur une produit défini pour être de qualité moyenne peut donc conclure au fait que les règles de contrôle qualité utilisées sont inutilement draconiennes !

Bien entendu, l'assurance qualité peut avoir pour objectif de vérifier le respect par un produit de certains niveaux de qualité. Ce besoin peut avoir été défini par le client, ou bien être le fruit d'une approche interne du développement. Boehm donne une liste (ouverte) de critères de qualité pouvant être pris en compte : économie, intégrité, documentation, compréhensibilité, flexibilité, interopérabilité, modularité, correction, fiabilité, modifiabilité, validité, généralité, testabilité, réutilisabilité, adaptabilité, utilisabilité, clarté, maintenabilité, portabilité, efficacité

L'assurance qualité s'appuie sur un ensemble de normes et de décisions appelées plan qualité. Il est souhaitable que le plan qualité soit défini aussi tôt que possible dans le projet, et décrive les standards appropriés au produit et au processus de développement. Notons que dès lors que des standards sont définis pour la rédaction des documents, l'assurance qualité doit les valider.

Lorsque l'assurance qualité impose des normes particulières, elle a la charge de leur diffusion auprès des personnels intéressés, sous forme de livres blancs ou d'extraits de normes et d'exemples.

7.1. l'assurance qualité du processus de développement

Dans certains processus industriels, la qualité requise du produit est atteinte au moyen d'itérations d'un processus de finition. On travaille en peaufinant le processus, jusqu'à avoir atteint la qualité voulue. En informatique, ce n'est pas le cas. Le processus de développement est découpé en phases bien définies, même si elles ne sont pas parfaitement consécutives sans recouvrement, et chaque phase doit produire un ensemble bien compris de documents.

L'assurance qualité intervient à ce niveau, en définissant des standards de développement (notamment la nature et le nombre des sous produits issus de chaque phase), en prévoyant des audits, ou revues, à dates fixes, et en effectuant des rapports.

Les audits constituent donc un moyen pour l'assurance qualité de garantir le respect du processus de développement déterminé au départ.

7.2. les standards définis par l'assurance qualité

Les normes imposées par l'assurance qualité déterminent les meilleurs pratiques, et donnent un support à sa mise en oeuvre. Ces standards sont définis de telle sorte que leur respect garantisse la réduction du coût de l'assurance qualité. Suivant les cas, il s'agira de formulaires, de règles de présentations, de règles de programmation etc...

L'assurance qualité rédige un "livre blanc", qui détermine les standards de la société sur la base des standards internationaux adéquats. Ce livre blanc est un document évolutif, car il doit suivre la pratique des équipes de développement, et ne jamais devenir désuet ou obsolète. Il faut dont permettre une remontée d'informations vers l'assurance qualité, notamment sous la forme de propositions d'évolutions des standards.

Notons enfin que les normes doivent être associés à des outils de contrôle, ou de mise en oeuvre automatique : formatage, indenteur. Par exemple, l'utilisation d'un indenteur de sources dispense (en théorie) les programmeurs d'indenter les programmes selon le modèle, mais garantit (en pratique) que tout le monde indente naturellement de la même manière ou presque.

7.3. les revues de qualité

7.3.1. objectif des audits

Une revue de qualité a pour objectif de déterminer les éventuelles erreurs, et de les indiquer au concepteur de l'élément faisant l'objet d'une revue. Ce dernier est responsable des éventuelles corrections, si la revue conclut à leur caractère impératif. On distingue deux types de revues suivant que l'intérêt se porte sur les programmes, ou sur le processus.

Dans le premier cas, les inspections de conception ou de programme ont pour objet la vérification de l'application des standards et contraintes, et le respect des spécifications préalables. Une telle revue vérifie donc la correction et la complétude d'un élément de code relativement à ce qui est demandé. On y utilise souvent une check list des erreurs les plus fréquentes.

Dans l'autre cas, les revues de gestion (ou d'avancement) ont pour objet l'évaluation du respect des contraintes du processus d'une part, et du respect des coûts, du planning, des délais d'autre part. Il s'agit d'un audit plus abstrait, ne nécessitant pas nécessairement d'étudier le code en détail.

7.3.2. équipe de la revue

Une revue comporte en général quatre à cinq personnes, soit extérieures au projet, soit issues du groupe ayant réalisé un produit[19]. Ces personnes sont responsables de la revue, dont le déroulement se fait sous la direction d'un président de séance appelé modérateur. Des experts, intervenants externes, ainsi que tous les membres de équipes de développement peuvent participer à la revue en émettant un avis circonstancié sur le document ayant circulé.

7.3.3. organisation de la revue

Dans une première phase, l'auteur du produit audité fait circuler son document aussi largement que nécessaire. Les commentaires de chacun doivent être consignés par écrit, et le modérateur doit garantir que tous les avis sont pris en compte ou bien classés non pertinents.

Ensuite, l'auteur fait un exposé à l'équipe de revue pour lui présenter son produit. Cette présentation ne dure pas généralement plus de deux heures, et est suivie d'un débat contradictoire.

Enfin, l'équipe de revue rédige un rapport, signé du président et de l'auteur du produit audité, où figurent des remarques (défaut ne nécessitant pas d'action correctrice), recommandations et demandes de corrections. Si la revue a fait apparaître un défaut très en amont (dans les spécifications par exemple), elle peut entraîner une cascade de réunions de mise au point.

Notons que le chef de projet peut appuyer sa gestion sur des délivrables "tamponnés" par l'assurance qualité, c'est à dire validés par une revue.

Egalement, dans certains projets, les revues ont un caractère contractuel : si le groupe d'audit avalise le produit qui lui est présenté, il engage sa responsabilité. Dans l'industrie aéronautique par exemple, les revues sont organisées par le consortium qui passe commande aux constructeurs. Si le produit présenté par un industriel est accepté lors d'une revue, il ne pourra plus contractuellement être rejeté comme non conforme sous la seule responsabilité de l'industriel. Il y a transfert de cette responsabilité[20]. Par exemple, une revue appelée "preliminary design review" doit attester que

[19]on entend ici "produit" au sens large : un docuement est un produit.

[20]en France (cas isolé en Europe), c'est dû au fait que les structures de l'état intègrent des hauts fonctionnaires très compétents techniquement.

le produit que chaque industriel se propose de réaliser traduit techniquement les exigences (contraintes) du contrat. Cette revue passée, le donneur d'ordre (le client) ne pourra plus contester la validité des choix techniques.